IRQ- ja NMI-keskeytykset. Taustagrafiikka. Spritet. Vieritys.
Musiikki ja äänitehosteet. Hyödyllisiä Kernalin rutiineita.
Erikoistehosteita videopiirillä.

BITTI -kirjat

C-64

PELINTEKIJÄN OPAS

Jukka Tapanimäki

RASTERI-KESKEYTYKSET

Commodore 64

Pelintekijän opas

Jukka Tapanimäki

Erikoislehdet Oy
Tecnopress

Commodore 64
Pelintekijän opas
Jukka Tapanimäki

© Erikoislehdet Oy, Tecnopress.
Kaikki oikeudet pidätetään.

Tämän teoksen tekstiosuuden osittainenkin kopioiminen on tekijänoikeuslain (404/61, muut. 897/80) ja valokuvalain (405/61, muut. 898/80) mukaisesti kielletty ilman kustantajan kirjallista suostumusta lukuun ottamatta tätä teosta koskevia arvosteluja ja katsauksia.

Kirjapaino Lars Eriksen Oy
Espoo 1990

ISBN 951-832-016-0

Sisältö

Johdanto 7
Ideasta peliksi 9
Toimintapelit 9
Seikkailupelit 10
Roolipelit 11
Strategiapelit 12
Muut peli-ideat 13
Ohjelman rakennuspalikat 14
Ohjelmoijan työkalut 14
Apuohjelmien käyttö 15
Hahmotetaan ongelma 16
Ohjelman rakenne 18
Nopeus vastaan tiiviys 19
IRQ- ja NMI-keskeytykset 20
Toiminta 20
Käyttöjärjestelmän IRQ-keskeytys 20
IRQ:n linkittäminen 21
Omat IRQ-rutiinit 21
Kontrollirekisteri 22
Ajastimien käyttö 24
NMI-keskeytykset 25
Vektorien käytöstä 26
Videopiirin rasterikeskeytykset 27
Rasterin rakenne 27
Rasterivertailu 27
Rasterikeskeytys 28
Keskeytysten vuorottelu 30
Harvinaisemmat keskeytystyypit 30
Rasterin ajoitus 31
Hienosäätöä 32
Taustagrafiikka 34
Värit 34
Videopankkien rakenne 35

Merkkigrafiikka 39
Moniväriset merkit 40
Laajennettu taustaväritila 41
Bittikarttagrafiikka 41
Osoitteen laskeminen bittikartalla 43
Bittikartan käyttäminen peleissä 45
Taustojen suunnittelu 45

Spritet 47

Spriten muoto 47
Sprite-lohkon valinta 48
Spriten sijoittaminen näytölle 49
Spriten värit 50
Spriten prioriteetit 51
Spriten laajentaminen 52
Törmäysten tutkiminen 52
Spriten monistaminen 52

Liikettä ruudulle 55

Liikkuvat spritet 55
Liikkuvat merkit 56
Pehmeyttä liikkeisiin 58
Liikettä bittikartalla 59

Vieritys 60

Vierityksen periaate 60
Nopeuden säätäminen 61
Kuvan kaventaminen 61
Vierityksen ongelmat 62
Vapaasuuntainen vieritys 63
Grafiikkaa muistista ruudulle 63
Esimerkki vaakavierityksestä 65

Musiikki ja äänitehosteet 70

Äänipiirin rakenne 70
Aaltomuoto 70
Verhokäyrä eli ADSR 72
Äänen korkeus 74
Äänen suodatus 74
Rengasmodulaatio ja synkronointi 75
Luettavat rekisterit 75
Digitoidut äänet 76

Peliohjaimen ja näppäimistön lukeminen 84
Joystick 84
Näppäimistö 86
Käytännön ongelmat 87
Hyödyllisiä Kernalin rutiineita 91
Tiedoston lataus 91
Tiedoston tallennus 92
Näppäimistön luku ja tekstin tulostus 93
Matematiikkaa 94
Negatiiviset luvut 94
Yhteen- ja vähennyslasku 95
Loogiset operaatiot: OR, AND ja EOR 97
Binaarikoodatut desimaaliluvut 98
Kertolasku 99
Jakolasku 101
Erikoisfunktiot 103
Erikoistehosteita videopiirillä 107
Reunusten avaaminen 107
Kuvan siirto pystysuunnassa 107
Kuvan siirto vaakasuunnassa 108
Liite 1
Muistin hallinta 117
Liite 2
Heksadesimaalit 119
Liite 3
Konekielikäskyt 120
Aakkosellinen hakemisto 129

Johdanto

Commodore 64 jää historiaan ensimmäisenä tietokoneena, josta tuli todellinen massatuote, "joka kodin kotimikro".

Aikansa kutakin. Tehokkaammat mikrot ovat vihdoin sysäämässä kuusnelosta syrjään. Konekanta on kuitenkin niin valtava, että kone ei katoa päivässä eikä kahdessa. Yksi kerrallaan ne jäävät syrjään upouuden supermikron vallatessa pöydän, työnnetään hyllyyn pölyttymään ja lopulta unohdetaan. Mutta vielä on konevaarissa henki sitkeässä.

Harrastajille kuusnelosesta on iloa vielä pitkään. Isompiin koneisiin verrattuna se on varsin helposti hallittavissa, minkä ansiosta se on mainio harjoitteluväline ja tirkistysikkuna konekielen ihmeelliseen maailmaan. Mikäli taidot ja itseluottamus riittävät, voi myös yrittää kaupallista tasoa olevien pelien ohjelmointia. Hyvälle pelille löytyy aina julkaisija, kunhan on tarpeeksi sinnikäs. Pienellä onnella pelillään voi ansaita 20 000 – 30 000 markkaa, mikä on toki vaivan arvoinen palkkio muutaman kuukauden työstä.

Tämän kirjan tarkoituksena ei ole opettaa ohjelmointia tai konekielen alkeita. Poikkeuksena on luku "Matematiikkaa", jossa perehdytään muutamiin tärkeisiin perusasioihin. Suurin osa kirjan sisällöstä keskittyy kuusnelosen esittelyyn peliohjelmoijan näkökulmasta. Niinpä etusijalla ovat keskeytysten hallinta, vieritys, taustagrafiikka, spritet ja muut peleissä oleelliset asiat. Kirja on myös mainio tuki ohjelmoijalle, joka haluaa ainoastaan rakennella näyttäviä demoja tai muuten vain tehdä löytöretkiä koneensa sisuksiin.

Ohjelmoinnin ei tarvitse olla tylsää ja vakavaa, ainakaan kuusnelosen ohjelmointi ei saisi olla tylsää ja vielä vähemmän vakavaa. Vietä siis hauskoja hetkiä koneesi parissa.

Sahalahdella 18. heinäkuuta 1990.

Jukka Tapanimäki

Ideasta peliksi

Kaikkiin peleihin tyylistä riippumatta pätee vanha ohje: on helpompi pitkästyttää kuin viihdyttää. Pelien suunnittelussa pulma on erityisen kipeä, sillä on hyvin vaikeaa löytää idea, jonka toteutuksessa ei joutuisi tyytymään kompromisseihin. Jo ennen ohjelmoinnin aloittamista edessä on mutkikas prosessi, joka on tarkkaa tasapainoilua nerokkaiden ideoiden ja teknisten rajoitusten välillä. Pelin ideointi on vähän kuin elokuvan tekoa: ensin tutkitaan käsikirjoitusta ja korvataan budjetin ylittävät kohtaukset yksinkertaisemmilla mutta yhtä tehokkailla jaksoilla.

Aivan ensimmäiseksi on selvitettävä itselleen pelin tyylilaji. Luonnollisesti valinta osuu sellaiseen pelityyppiin, josta itsekin pitää ja johon uskoo omien taitojensa riittävän. Rimaa ei saisi asettaa liian korkealle eikä myöskään liian matalalle, sillä kummassakin tapauksessa ohjelmointiin saattaa kyllästyä kesken kaiken. Puoliksi tuttua rutiinityötä, puoliksi haastetta, siinä hyvä resepti, jolla pidetään motivaatiota yllä.

On hyvä tietää millaisiin ongelmiin erityyppisiä pelejä ohjelmoidessaan tulee törmäämään. Seuraavassa on lueteltu pohdiskelun arvoista pohjatietoa.

Toimintapelit

Kuusnelosen grafiikkapiiri mahdollistaa kuvaruudun pehmeän vierityksen ja helppokäyttöiset spritet, joiden ansiosta kone soveltuu hyvin toimintapelien ohjelmointiin. Ei silti kannata tuudittautua uskoon, että kone tekee kaiken ohjelmoijan puolesta.

Toimintapelit saavat varsin usein alkunsa jostain yksinkertaisesta rutiinista. Kuusnelosella tällainen lähestymistapa onkin järkevä, sillä on hyvä saada ensin tuntuma koneen tehosta ja vasta sitten sännätä ideoimaan uutta supersarjan räiskintäpeliä. Tyypillinen esimerkki on vieritys: ohjelmoija tekee kokeilumielessä vieritysrutiinin, joka osoittautuu siistiksi ja tehokkaaksi. Vasta sitten mieleen hiipii ajatus, että sitä voisi hyödyntää pelissä.

Mitä toiminnallisempi peli, sitä enemmän joutuu painiskelemaan teknisten ongelmien parissa. Pelihahmojen liikkeiden sujuvuus, ohjaustuntuma ja lukemattomat muut pienet yksityiskohdat vaativat paljon kärsivällistä hiomista. Jotta lopputulos olisi siisti, koko ruudun sisältö pelihahmoineen ja taustoineen joudutaan piirtämään viisikymmentä kertaa sekunnissa. Tässä hommassa kuusnelonen joutuu usein niin koville, että aika ei kerta kaikkiaan riitä. Niinpä ohjel-

moinnin edistyessä on keksittävä yhä uusia niksejä, joilla aikakriittisiä rutiineita voisi nopeuttaa. Yleensä nopeuden lisääminen lisää muistin kulutusta, toisaalta taas pelin taustagrafiikka olisi sullottava mahdollisimman tiiviiksi. Näiden välillä tasapainoilu onkin jo taito sinänsä.

Toimintapelin ohjelmoija sortuu helposti tekniseen kikkailuun. Peli on näyttävä, mutta ideointi on jäänyt sivuseikaksi. Pelintekijä keskittyy viilaamaan koodiaan ja lopuksi joutuu toteamaan pettyneenä, että kukaan ei arvosta hänen hienoja ohjelmointiniksejään. Kaupallisen pelin tekeminen on aivan eri asia kuin vaikutuksen tekeminen ohjelmoinnista kiinnostuneeseen tuttavapiiriin. On muistettava, että suuri osa pelaajista ei ymmärrä ohjelmoinnista mitään.

Toimintapelin ideaksi ei missään tapauksessa saisi riittää se, että näytöllä pyörii avaruusalus, joka ampuu kentän tyhjäksi vihollisista. Sen sijaan tämän perusidean ympärille voi rakennella aivan mitä tahansa. Pelistä pitää loistaa läpi jokin, vaikka miten pieni omaperäisyyden kipinä. Turhaa kikkailua on silti vältettävä ja yritettävä keskittyä olennaiseen: pelattavuuteen. Pelaa paljon sellaisia toimintapelejä, joista pidät, tutki mikä tekee ne miellyttäväksi pelata ja sen jälkeen yritä keksiä, miten saman voisi tehdä vieläkin paremmin.

Seikkailupelit

Pelkästään tekstiin perustuvalla seikkailupelillä nykyisin tuskin pääsee kaupan hyllylle asti, mutta niiden suunnittelu ja ohjelmointi on hyvää harjoitusta, varsinkin kun Basicillakin voi yltää tyydyttävään tulokseen. Erityisen opettavaista on konekielisen osuuden lisääminen asteittain. Esimerkiksi konekielinen pareseri mahdollistaa laajan sanavaraston ja mutkikkaat lauserakenteet, jolloin peleistä voi tehdä yhä laajempia ja sujuvammin toimivia.

Paljon houkuttelevampi vaihtoehto kaupallisesti suuntautuneelle koodaajalle ovat graafiset seikkailupelit, jotka ovat saavuttaneet laajan suosion. Päältä katsottuna pelit näyttävät yksinkertaisilta, mutta niiden ohjelmointi on kaikkea muuta. Esimerkiksi pelihahmon liikkuminen kolmiulotteisesti kuvatuissa maisemissa Sierran pelien tyyliin vaatii tavattoman joustavaa ja hyvin suunniteltua ohjelmaa. Parhaimmillaan pelin rungon ohjelmointi hipoo mutkikkuudessaan käyttöjärjes-

telmän suunnittelua. Kannattaa myös pitää mielessä järkyttävä muistin kulutus: useamman megatavun kokoiset seikkailupelit eivät ole mitenkään harvinaisia.

Ohjelmoijan kannalta graafisten seikkailupelien suurin etu on siinä, että kun ohjelmoinnista on kerran selvinnyt, samaa koodia voi käyttää pohjana lukemattomissa uusissa peleissä. Juuri näin tekevät seikkailun jättiläiset Lucasfilm ja Sierra.

Käytti pelissään sitten tekstiä tai grafiikkaa, suurin ongelma on sittenkin pelin juonen ja sisäisen logiikan rakentelu. Juonen on oltava omaperäinen, mahdollisesti hieman omalaatuinen, miksei myös täysin pähkähullu. Tärkeintä on saada pelaaja kiinnostumaan. Vielä parempi, jos joku jaksaa pelata seikkailun loppuun asti, mutta todennäköisesti kukaan ei sitä tee, jos ratkaistavaksi asetetut ongelmat on suunniteltu huonosti. Kiinnostavien ja sopivan vaikeiden ongelmien suunnittelu on vaikeasti opittava taito. Se vaatii keskivertoa enemmän älyä ja kekseliäisyyttä.

Roolipelit

Roolipelit ovat häilyvärajainen ryhmä, jossa ainoa yhdistävä piirre on sankareiden ominaisuuksien muuttuminen pelin kuluessa. Tekninen toteutus on usein yhtä häilyvää sijoittuen jonnekin toiminta-, seikkailu- ja strategiapelien välimaastoon.

Yksi roolipelien ohjelmoinnin suurimmista ongelmista on koodin pirstaleisuus. Pelissä voi olla useita hyvinkin erilaisia jaksoja: pelimaailman tutkimista, taistelemista, keskustelemista outojen olentojen kanssa jne. Mikään näistä ei yksinään tuota ylivoimaisia ongelmia, mutta osien punominen yhteen on vaativampi tehtävä.

Toinen hankaluus on roolipelien taipumus paisua paisumistaan. Kuusnelosen muisti loppuu kesken yllättävän nopeasti, kun innokas ohjelmoija pääsee vauhtiin. Tärkein tavoite onkin koodin puristaminen mahdollisimman tiiviiksi, silläkin uhalla, että lopputulos vaikuttaa hieman kömpelöltä verrattuna hyvin ohjelmoituun toimintapeliin.

Roolipelien kuten myös seikkailupelien kohdalla kunnianhimoisella koodaajalla on väkisinkin edessään levyaseman käytön opiskelu. Massamuistin käyttö lisää entisestäänkin ohjelman pirstaleisuutta ja ohjelman hallitseminen käy yhä vaikeammaksi. Ratkaistavana on myös uusia käytännön ongelmia. Mitkä osat pelistä kannattaisi ladata levyltä? Missä muodossa ne on sinne tallennettu? Pitäisikö tiedostojen editoimista varten tehdä erillisiä apuohjelmia? Pulmat ovat sen verran kiperiä, ettei aivan ensikertalaisen kannata haaveilla Ultiman tasoisen roolipelin ohjelmoinnista, ei ainakaan puhtaalla konekielellä.

Monimutkaisuudestaan huolimatta, tai ehkä juuri siitä johtuen roolipelien suunnitteleminen ja ohjelmointi on hyvin antoisaa puuhaa. Tehtävänä on luoda kokonainen toimiva maailma, joka on tulvillaan sankarihahmoja, magiaa, seikkailuita ja mieltä kiehtovia mysteereitä. Pelaajalle on tarjottava seikkailuareenaksi paikka, joka on yhtä aikaa uskottava, tarunomainen ja mahdollisimman kaukana tylsästä arjesta. Eipä siis ihme, että roolipelien suosituimpia tapahtumapaikkoja ovat myyttinen menneisyys ja kaukainen tulevaisuus.

Strategiapelit

Strategiapelit ovat perinteisesti olleet vähiten tekninen pelityyppi. Usein niissä on tyydytty karuun lautapelimäiseen ulkoasuun, mutta mikään ei tietenkään estä käyttämästä hyvinkin pitkälle kehiteltyjä ohjelmointitekniikoita, kuten vaikka kolmiulotteisia fraktaalimaisemia.

Strategiapelit eroavat ratkaisevasti edellä mainituista pelityypeistä siinä, että ne vaativat ehdottomasti älykkään tietokonevastustajan. Parhaaseen tulokseen päästään, jos pelissä on useita älyltään vaihtelevia vastustajia.

Alkeellisenkin tekoälyn ohjelmointi on vaativa tehtävä. Tietokoneen olisi pystyttävä arvioimaan pelitilanne ja suunnittelemaan oma taktiikkansa tämän tiedon pohjalta. Koneen olisi pystyttävä näkemään tulevaisuuteen muutaman siirron verran, minkä seurauksena vähänkin mutkikkaammassa pelissä koneen miettimisaika voi venähtää kohtuuttomaksi. Ongelmana on myös konepelaajan kaavamaisuus. Sen siirrot on liian helppo ennakoida ja pelaaminen käy tylsäksi. Niinpä tekoälyä olisi syytä hieman "inhimillistää", toisin sanoen konepelaajan pitäisi pystyä harkittuihin riskeihin, hämäykseen ja jopa tahallisiin virheisiin.

Jos tehtävä tuntuu liian vaikealta, voi tietysti tehdä kahden pelattavan strategiapelin, mutta tällöin tuloksena on pelkästään tietokoneella pelattava seurapeli, ei varsinainen tietokonepeli.

Muut peli-ideat

Pelien karkeaa perusjaottelua voi täsmentää lukemattomilla määritelmillä. Esimerkiksi toimintapeleistä löytyy alaosasto toimintaseikkailut, joista edelleen voidaan eristää omaksi ryhmäkseen isometrista viistoperspektiiviä käyttävät pelit. Ehkä selvimmin muista eroavia pelityyppejä ovat simulaatiot ja ongelmapelit.

Simulaatiossa pyritään jäljittelemään todellisuutta, oli se sitten tikanheittoa tai avaruusmatkailua. Hyvässä simulaatiossa pitäisi ottaa huomioon lukematon määrä tekijöitä, jotka eivät välttämättä ole edes kiinnostavia lopputuloksen kannalta. Kuusnelosella ohjelmoitaessa on parempi tyytyä "simulaation näköiseen" peliin ja keskittyä pelattavuuteen. Esimerkiksi urheilupelit voi luokitella eräänlaisiksi simulaatioiksi.

Ongelmapelit ovat epämääräinen ryhmä pelejä, joissa pelaajalta vaaditaan älyä ja oivallusta pelkän reaktiokyvyn sijaan. Strategiapeleistä ongelmapelit eroavat sikäli, että niissä on toiminnallisempi ote. Ne on yleensä erittäin helppo ohjelmoida, mutta yksinkertaisistakin aineksista saa nautittavan pelin. Tetris on tästä hyvä esimerkki. Suurin hankaluus onkin omaperäisen perusidean keksiminen.

Perinteinen jaottelu on tietysti hyvin keinotekoinen, sillä peliin on mahdollista sekoittaa useita tyylejä ja teknisiä ratkaisuita. Ei kuitenkaan kannata olla suuruudenhullu. Yksi idea hyvin toteutettuna on parempi kuin löyhästi yhteen nivottu möhkäle puolittain toimivia pirstaleita.

Ohjelman rakennuspalikat

Pelissä on tärkeintä lopputulos. Periaatteessa on yhdentekevää millaisin välinein ohjelma on tehty ja millaista koodia ohjelmoija syytää, kunhan peli toimii. On kuitenkin hyvä hallita muutamia perusniksejä, jos ei muuten, niin ainakin oman mielenterveytensä takia.

Ohjelmoijan työkalut

Konekielisen ohjelman tekemisessä tarvitaan kaksi apuvälinettä, konekielimonitori ja assemblerkääntäjä.

Konekielimonitorin avulla on helppo tutkia muistissa olevaa konekielistä ohjelmaa, sillä samalla kun muistia selataan, se muuttaa konekieliset käskyt assemblerkoodeiksi. Monitori on myös eräänlainen yksinkertaistettu assemblerkääntäjä, jolla ohjelman voi kirjoittaa suoraan koneen muistiin.

Ohjelmointivälineenä konekielimonitori ei ole käytännöllinen, sillä ohjelmaa on hyvin vaikea editoida. Esimerkiksi kahden käskyn väliin ei voi lisätä mitään ennen kuin ohjelman koko loppuosaa on siirretty muistissa ylöspäin ja samalla muutettu kaikkien hyppykäskyjen osoitteita. On suositeltavaa, että konekielimonitoria käytetään vain ohjelman testaamiseen tai hyvin lyhyiden rutiinien kirjoittamiseen. Ohjelman kasvaessa sen muuttaminen käy lopulta aivan liian työlääksi.

Konekielimonitoria on kätevintä käyttää, jos se on sijoitettu erilliselle moduulille. Tällöin se ei vie lainkaan koneen muistia, eikä näin ollen ole myöskään ohjelmoijan tiellä. Useat moduulit on suunniteltu siten, että monitorin voi käynnistää melkein milloin tahansa yhdellä napin tai näppäimen painalluksella.

Assemblerilla ohjelmoitaessa koodi tallennetaan symboliseen muotoon, josta se käännetään varsinaiseksi konekieliohjelmaksi. Tämän ansiosta sitä voi editoida samaan tapaan kuin Basic-ohjelmaa: leikellä, lisäillä, siirtää palasia eri kohtaan. Joustavuuden lisäksi etuna on symboleiden käyttö, esimerkiksi muistipaikkoja voi käyttää muuttujina samaan tapaan kuin Basicissa.

Kuusnelosen konekielikääntäjistä ehkä laajimmalle levinnyt on turboassembler. Sen etuna on helppokäyttöisyys ja nopeus. Ohjelma käännetään suoraan muistiin, minkä ansiosta sitä pääsee heti testaamaan. Haittana taas on ohjelman koon rajallisuus. Se on pakko saada mahtumaan sille rajattuun tilaan, tai jotain on jätettävä pois. Usein tarvitaan niin paljon grafiikkaa ja taulukoita, että ne on

sijoitettava assemblerin päälle, jolloin nopeudesta saatu etu karisee tiedostojen jatkuvaa latailuun.

Laajempiin ohjelmiin sopii paremmin assembler, joka tekee ohjelmasta erillisen tiedoston levylle. Tällöin ohjelma voi käyttää koko muistin. Assemblerkoodi vie kuitenkin paljon enemmän muistia kuin konekieli, suunnilleen viisinkertaisen määrän, minkä vuoksi vähänkin pitemmän ohjelman joutuu pilkkomaan osiin. Ne liitetään yhdeksi kokonaisuudeksi vasta käännösvaiheessa.

Ohjelmoijan apuvälineistä tehokkain on ristiinkääntäjä. Ristiinkääntämisellä tarkoitetaan sitä, että käännettäessä ohjelma konekieliseksi se samalla siirretään toiseen tietokoneeseen. Ristiinkääntämiseen voi käyttää kahta kuusnelosta, mutta on paljon parempi käyttää ohjelman kirjoittamiseen tehokkaampaa mikroa, jolloin kääntäminen on nopeampaa ja assemblerohjelma mahtuu kerralla tietokoneen muistiin. Esimerkiksi tämän kirjan rutiinien kirjoittamisessa ja testauksessa on käytetty englantilaisen PD Systemsin ristiinkääntäjää, joka toimii PC-mikrossa.

Apuohjelmien käyttö

Pelkällä assemblerilla peliohjelmoija tuskin pärjää, varsinkin jos pelissä on runsaasti grafiikkaa. Periaatteessa grafiikan voi rakennella ruutupaperilla ja muuttaa sen sitten numerokoodeiksi, mutta se alkaa nopeasti tuntua ylivoimaisen työläältä. On paljon mukavampaa käyttää erillisiä työkaluohjelmia.

Pelinteko-ohjelma Shoot'em up Construction Kit on esimerkki hyvin tehdystä apuohjelmasta. Siinä on mukana erinomaiset editorit spriteja, grafiikkamerkkejä ja pelin taustoja varten. Valitettavasti ohjelman kyvyt rajoittuvat pystyyn

vieriviin ammuskelupeleihin, joiden tekeminen ei pitkään jaksa kiinnostaa. Mutta tuskin virkavalta tulee koputtamaan ovellesi, jos käytät ohjelmaa pelkästään grafiikan luomiseen omiin peleihisi. On silti kohteliasta erikseen mainita, mitä kaupallista apuohjelmaa grafiikan luomiseen on käytetty.

Parhaat apuohjelmat ovat itse tehtyjä. Niitä osaa varmasti käyttää ja niihin voi myös lisäillä uusia ominaisuuksia tarpeen vaatiessa. Spriteja ja grafiikkamerkkejä varten riittää yksi yleiskäyttöinen editori. Sen sijaan taustagrafiikan voi rakentaa niin monella eri tavalla, että erilaista peliä varten tarvitaan myös uusi työkaluohjelma.

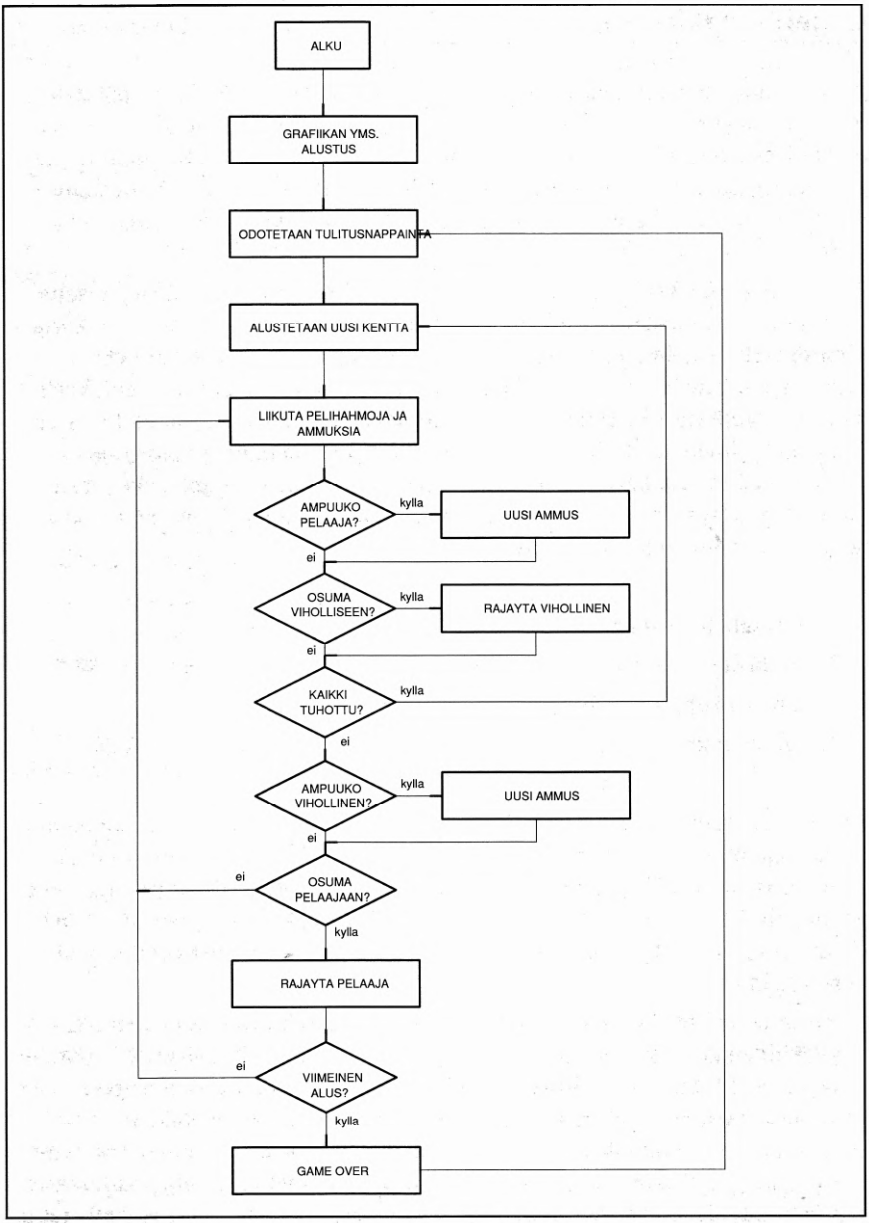
Hahmotetaan ongelma

Ennen ohjelmoinnin aloittamista kannattaisi lukita kuusnelonen tallelokeroon ja postittaa avain itselle pikakirjeenä (tällä tavoin se viipyy matkalla ainakin viikon). Sitten vain mietiskellään peli-ideaa teetä juodessa, golfia pelatessa, porealtaassa istuskellessa tai missä mieli tekee.

Vuokaavioiden ja muiden rautalankamallien käyttö on makuasia, tärkeintä on saada jonkinlainen kuva kokonaisuudesta, osien tärkeysjärjestyksestä ja ohjelmointiin liittyvistä ongelmista. Hyvän esityön jälkeen palat loksahtavat paikoilleen kuin itsestään. Yleensä koneeseen ei kannata edes koskea ennen kuin seuraavat kriteerit on täytetty:

1. Idea on tarpeeksi innostava ja motivoi ohjelmoijaa.
2. Idea on loppuun asti mietitty – ainakin runko.
3. Sekä koneen teho että omat taidot riittävät.
4. Pelin toiminnasta on selkeä kokonaiskuva.

Kohta 2 tarkoittaa, että aivan kaikkea on turha tai peräti mahdoton päättää ennakolta. Hämmästyttävän usein parhaat ideat tulevat vasta kun ohjelmointi on jo vauhdissa. On jopa varauduttava siihen, että lopputuloksessa ei ole mitään jäljellä alkuperäisestä suunnitelmasta. Pelien ohjelmointi on luovaa toimintaa, joten ohjelmassa on jatkuvasti oltava hieman kasvuvaraa joka suuntaan, sillä koskaan ei tiedä milloin päähän pälkähtää uusi nerokas oivallus. Tässä suhteessa pelinteko eroaakin ratkaisevasti systeemisuunnittelusta ja muusta "eksaktista" ohjelmointityöstä.



Kaavio 2.1. Vuokaaviolla voi karkeasti hahmotella ohjelman toiminnan. Kaavio voisi kuvata esimerkiksi Space Invaders -peliä.

Ohjelman rakenne

Aloittelijalle on tyypillistä vääntää koodia väkisin. Rynnätään kylmäpäisesti koneen ääreen ja ratkotaan ongelmat sitä mukaa kun niihin törmätään. Lyhyissä ohjelmissa voi selvitä ilman suunnittelua, mutta ei täysimittaisessa pelissä. Jossain vaiheessa ohjelma on yleensä niin solmussa, että sen laajentaminen käy yhä työläämmäksi, kunnes lopulta tekijä itsekin on pudonnut täydellisesti kärryiltä.

Kannattaa omaksua mahdollisimman siisti ohjelmointityyli. Akateemiselta hienostelulta kuulostava termi "rakenteinen ohjelmointi" tarkoittaa yksinkertaisesti sitä, että ohjelma jaetaan alirutiineihin ja kussakin tehdään vain yksi selkeästi rajattu tehtävä. Peliohjelmoinnissa jaottelu on niin luontevaa, että kokemuksen myötä sitä ei tarvitse enää edes miettiä: pelihahmojen liikuttelu, vieritys, peliohjaimen lukeminen ja niin poispäin. Erityisen mutkikkaissa operaatioissa kannattaa kaivaa esiin kynä ja paperia ja purkaa ongelma kevyesti pureskeltavaksi paloiksi. Esimerkiksi ongelman "heitämme keihästä tietokoneella" voi hahmotella seuraavasti:

1. Juoksuta urheilijaa.
2. Säädä heittokulma.
3. Lennätä keppiä heittokulman ja -nopeuden mukaisesti.
4. Töksäytä keppi maahan, laske tulos ja piristeeksi suosionosoituksia.

Karkean jaottelun jälkeen voi keskittyä yhteen kohtaan kerrallaan ja pohtia millaisia ongelmia niihin liittyy. Miten juoksunopeus säädetään? Miten keihään lento on kuvattu? Mitä tapahtuu, jos heitto on yliastuttu? Entä jos pelaaja tekee jotain hölmöä, esimerkiksi kohdan 2 jälkeen ei päästäkään keppiä kädestään? Yksinkertaistakin ohjelmointitehtävää purettaessa voi törmätä hyvinkin yllättäviin ongelmiin.

Alirutiinien käyttämisestä on kahdenlaista hyötyä. Ensinnäkin ohjelman muuttaminen ja lisäysten tekeminen on helpompaa, kun tehtäviä ei ole ripoteltu sinne tänne pitkin koodia. Siisti ohjelma on kuin kortisto, jossa on helppo vaihtaa operaatioiden järjestystä ja vaikkapa poistaa jokin ominaisuus testauksen ajaksi, esimerkiksi törmäysten tutkiminen. Toinen ja ainakin yhtä tärkeä seikka on virheiden korjaamisen helppous. Jos ohjelma ei tuntemattomasta syystä toimi, alirutiinit voi eristää yksi kerrallaan kunnes virhe on paikallistettu.

Jokaista rutiinia ohjelmoitaessa olisi pidettävä mielessä, että sen laajentamisen pitäisi olla mahdollisimman helppoa. Jos pienikin lisäys vaatii ohjelman

muuttamista kymmenessä kohdassa, jotain on mennyt pieleen. Ohjelman tekeminen joustavaksi vaatii ylimääräistä vaivaa, mutta myöhemmin se palkitaan moninkertaisesti.

Nopeus vastaan tiiviys

Hyvän ohjelmoijan koodi on sekä tiivistä että nopeaa. Valitettavasti tähän kätkeytyy sellainen ristiriitaisuus, että tiiviissä ohjelmassa muistin kulutus on itse asiassa suurempaa, sillä ohjelma vaatii tuekseen paljon pitkiä taulukoita.

Tutkitaanpa esimerkiksi rutiinia, jossa lasketaan spritea vastaava bitti tiettyjä grafiikkarekistereitä varten:

```
ESIM1   LDA #1
        LDX SPRITE
        BEQ END
LOOP    ASL A
        DEX
        BNE LOOP
END     RTS
```

Akussa olevaa bittiä siis siirrettiin spriten järjestysnumeron mukaisesti. Taulukkoa käyttämällä saman asian voi tehdä tehokkaammin:

```
ESIM2   LDX SPRITE
        LDA SPRBIT,X
        RTS
SPRBIT.BYTE 1,2,4,8,16,32,64,128
```

Jälkimmäinen rutiini on huimasti nopeampi, mutta muistiakin kuluu enemmän, tosin tässä tapauksessa eroa on vain muutama tavu. Parempia esimerkkejä ovat kuvaruutuosoitteen laskeminen bittikartalla, trigonometriset funktiot ja muut vastaavat. Taulukot voivat paisua satojen tavujen pituisiksi, kun taas ohjelmakoodi typistyy murto-osaan.

Operaation luonteesta riippuu, onko nopeus muistin kulutusta tärkeämpää. Esimerkiksi grafiikan liikuttelussa ei pahemmin kannata pihtailla muistin kanssa. Toisaalta taas tekstin kirjoittamista ruudulle on aivan turha tehdä turboviritellyllä koodilla.

IRQ- ja NMI-keskeytykset

Ohjelmoija kohtaa usein tilanteita, joissa olisi suunnattomasti apua, jos tiettyjä alirutiineita voisi kutsua tasaisin välein. Näppäimistön lukeminen ja kuvaruudun vieritys ovat tästä hyviä esimerkkejä. Keskeytykset ovat ratkaisu ongelmaan. Prosessorin sisään on rakennettu järjestelmä, jonka avulla haluamansa alirutiinit voi suorittaa tasaisin välein muun ohjelman toiminnasta riippumatta. Keskeytys pysäyttää pääohjelman toiminnan ja ohjelman suoritus jatkuu keskeytysrutiinissa. Sen jälkeen palataan takaisin pääohjelmaan aivan kuin mitään ei olisi tapahtunutkaan.

Toiminta

Keskeytysten toiminta perustuu keskeytyslähteen prosessorille antamaan keskeytyspyyntöön. Yleisimmin käytettyjä keskeytyslähteitä ovat ajastimet, joista tarkemmin vähän myöhemmin, ja rasterivertailu, johon tutustutaan seuraavassa luvussa.

Kun prosessori saa pyynnön, se automaattisesti tallentaa pinoon kolmen tavun verran tietoa: keskeytyskohdan osoitteen järjestyksessä ylä- ja alatavu sekä tilarekisterin arvon. Tämän jälkeen ohjelman suoritus siirtyy keskeytysrutiiniin. Keskeytyksen tyypistä riippuu, mistä prosessori löytää rutiinin osoitteen.

Keskeytysrutiinin on päätyttävä RTI-käskyyn, jonka toiminta on käänteinen keskeytyspyynnölle. Pinosta luetaan ensin tilarekisteri ja sitten osoite, josta keskeytettyä ohjelmaa jatketaan.

Käyttöjärjestelmän IRQ-keskeytys

Basic-tulkkia käytettäessä kuusnelosessa on käytössä tavallinen IRQ-keskeytys (lyhenne sanoista Interrupt Request). Keskeytysrutiinia kutsutaan noin 60 kertaa sekunnissa ja sen tehtävänä on tutkia näppäimistöä, päivittää sisäistä kelloa, vilkuttaa kursoria yms.

Rutiiniin hyppääminen tapahtuu siten, että prosessori lukee osoitteista $FFFE ja $FFFF hyppyvektorin, jossa keskeytysrutiinin osoite, normaalisti $FF48, on järjestyksessä ala- ja ylätavu. Rutiinissa tallennetaan akku, X- ja Y-rekisteri pinoon ja tutkitaan onko kyseessä konekielisen BREAK-käskyn aiheuttama keskeytys. Jos BREAK-lippu on päällä, luetaan uusi hyppyvektori

osoitteista $0316 ja $0317. Muussa tapauksessa luetaan osoitteista $0314 ja $0315 hyppyvektori tavalliseen keskeytysrutiiniin. Normaalisti se alkaa osoitteesta $EA31.

IRQ:n linkittäminen

Helpoin tapa käyttää keskeytyksiä on oman alirutiinin linkittäminen käyttöjärjestelmän valmiiseen rutiiniin. Se onnistuu esimerkiksi seuraavalla koodinpätkällä:

```
LINK    SEI             ;keskeytykset pois
        LDA #<IRQ       ;oman rutiinin osoitteen alatavu
        STA $0314
        LDA #>IRQ       ;oman rutiinin osoitteen ylätavu
        STA $0315
        CLI             ;keskeytykset päälle
        RTS             ;palataan Basiciin
                        ;tai omaan konekielirutiiniin
```

Linkitetyssä alirutiinissa ei ole muuta erikoista kuin että sen on päätyttävä käskyyn JMP $EA31, jotta keskeytys tulisi käsiteltyä asiallisesti.

Keskeytysten estäminen SEI-käskyllä on välttämätöntä, sillä on mahdollista, että keskeytys tapahtuu juuri kun vektorista on muutettu vasta ensimmäinen tavu.

Omat IRQ-rutiinit

Käyttöjärjestelmän keskeytysten hyödyntäminen ei ole kovin taloudellista. Jos ei tarvita näppäimistöä, ei ole mitään järkeä lukea sitä 60 kertaa sekunnissa. Mieluummin kannattaa kirjoittaa koko keskeytysrutiini itse. Tällöin Kernal on kytkettävä pois päältä, jotta hyppyvektorin voi muuttaa suoraan osoitteissa $FFFE ja $FFFF. Keskeytyksen alustuksessa on hieman eroa edellä listattuun LINK-rutiiniin:

```
OMAIRQ SEI
       LDA #$35     ;Kernal pois päältä
       STA $01
       LDA #<IRQ    ;muutetaan IRQ-vektori
       STA $FFFE
       LDA #>IRQ
       STA $FFFF
       CLI
       RTS          ;kutsuttava JSR-käskyllä,
                    ;ei Basicin SYS-komennolla!
```

Keskeytysrutiinissa on nyt huolehdittava itse keskeytyksen käsittelystä. On muistettava tallentaa rekisterit, kuitattava keskeytys ja päätettävä rutiini RTI-käskyyn:

```
IRQ    STA $FB      ;rekisterit talteen
       STX $FC
       STY $FD
       JSR OMA      ;kutsutaan omaa alirutiinia
       LDA $DC0D    ;kuittaus (myös BIT $DC0D käy)
       LDA $FB      ;palautetaan rekisterien arvot
       LDX $FC
       LDY $FD
       RTI          ;paluu keskeytyksestä
```

Rekisterit voi tietysti tallentaa myös pinoon, mutta nollasivun käyttäminen on tehokkaampi ja myös havainnollisempi tapa. Keskeytyksen kuittaamiseen käytettävä turhantuntuinen käsky LDA $DC0D on välttämätön, sillä ilman sitä CIA-piiri luulee, ettei keskeytystä ole vielä käsitelty ja prosessori saa välittömästi uuden keskeytyspyynnön.

Kontrollirekisteri

Complex Interface Adapterissa eli CIA-piirissä on rekistereitä, joilla keskeytysten toiminnan voi säätää mieleisekseen. Taulukossa 3.1 on lueteltu keskeisimmät rekisterit ja niiden tärkeimmät bitit.

$DC04	Ajastin A, alatavu
$DC05	Ajastin A, ylätavu
$DC06	Ajastin B, alatavu
$DC07	Ajastin B, ylätavu
$DC0D	Keskeytysten kontrollirekisteri
	Bitti 7: 0 = nollaa bitit, 1 = kirjoita bitit
	Bitti 1: 1 = keskeytyslähteenä ajastin B
	Bitti 0: 1 = keskeytyslähteenä ajastin A
$DC0E	Kontrollirekisteri A
	Bitti 4: 1 = "force load" eli ajastin A pakotetaan alkuarvoon
	Bitti 3: 0 = jatkuva, 1 = vain yksi kierros ajastimella A
	Bitti 0: 0 = pysäytä, 1 = käynnistä ajastin A
$DC0F	Kontrollirekisteri B
	Bitit 5-6: 00 = laske kellosyklejä (normaali)
	10 = laske ajastimen A kierroksia
	Bitti 4: 1 = "force load" eli ajastin B pakotetaan alkuarvoon
	Bitti 3: 0 = jatkuva, 1 = vain yksi kierros ajastimella B
	Bitti 0: 0 = pysäytä, 1 = käynnistä ajastin B

Taulukko 3.1. CIA #1 -piirin keskeytyksiin liittyvät rekisterit ja tärkeimmät bitit. CIA #2:n vastaavissa rekistereissä osoitteena on $DD00–.

Muistipaikassa $DC0D on kontrollirekisteri, jolla määrätään keskeytyslähde. Rekisterin toiminta on sikäli erikoista, että ylin eli bitti seitsemän määrittelee miten muut rekisteriin kirjoitettavat bitit tulkitaan. Jos bitti on nolla, kirjoitettavat ykkösbitit nollaavat rekisterin bittejä. Jos taas bitti on ykkönen, muut bitit kirjoitetaan rekisteriin. Ylintä bittiä lukuunottamatta nollabitit eivät vaikuta mitenkään rekisterin sisältöön.

Kirjoitettaessa rekisteriin $7F (binaarina %0111111) rekisterin kaikki bitit nollataan. Kirjoitettaessa $81 (binaarina %1000001) pistetään päälle rekisterin ensimmäinen bitti, joka määrittelee keskeytyslähteeksi A-ajastimen. Vastaavasti luvulla $82 (binaarina %10000010) keskeytyslähteeksi saadaan B-ajastin.

Rekisteriin kirjoittaminen ja sen lukeminen ovat kaksi täysin eri asiaa. Sen arvo poikkeaa nollasta vain keskeytyspyynnön yhteydessä ja keskeytyslähteen kertova bitti pysyy päällä kunnes se luetaan. Niinpä onkin mahdollista käyttää kumpaakin ajastinta samaan aikaan kirjoittamalla rekisteriin $83 (binaarina

%10000011), koska keskeytysrutiinissa on mahdollista tutkia kummasta ajastimesta keskeytys on lähtöisin.

Ajastimien käyttö

Ajastimet ovat 16-bittisiä rekistereitä, joilla määritellään keskeytysten tiheys. "Aikayksikkönä" on prosessorin kellotaajuus, mikä on rahtusen verran alle yhden mikrosekunnin. Normaalisti keskeytyslähteenä on A-ajastin ja tiheydeksi on määritelty $4025 (= 16421) kellojaksoa. Pienellä taskulaskimen näpelöinnillä selviää, että taajuus on todellakin 1/60 sekuntia, kuten aiemmin todettiin.

Peliohjelmoijan kannalta keskeytysten normaali taajuus on hieman epämukava, sillä kuvan päivitystaajuus on hieman hitaampi, 1/50 sekuntia. Ajastimen arvolla $4CC7 taajuuden saa synkkaan kuvaruudun kanssa. Listauksessa on esimerkki, miten alustetaan IRQ-keskeytys, joka "simuloi" videopiirin rasterikeskeytystä:

```
FREQ50 SEI
       LDA #$35
       STA $01
       LDA #<IRQ
       STA $FFFE
       LDA #>IRQ
       STA $FFFF
       LDA #$C7     ;ajastin A = $4CC7
       STA $DC04
       LDA #$4C
       STA $DC05
       LDA #$81     ;keskeytyslähteeksi ajastin A
       STA $DC0D
       LDA #$FB
WAIT   CMP $D012    ;odotetaan rasteririviä $FB
       BNE WAIT
       LDA #$11     ;ajastimeen "pakkosyötetään"
                    ;edellä määritelty arvo
       STA $DC0E
       CLI
       RTS
```

Nyt keskeytykset osuvat aina rasteririvin $FB (251) paikkeille, mikä on juuri kuvaruudun näkyvän alueen alla. Huomaa että ajastimen kontrollirekisteriin $DC0E kirjoitettavassa luvussa on päällä bitit 0 ja 4. Nollabitillä käynnistetään ajastin, yleensä se on jo valmiiksi päällä. Nelosbitin toiminta onkin erikoisempaa: sen avulla ajastimeen pakotetaan arvo, joka sille on määritelty taajuudeksi. Muussa tapauksessa ajastin laskee kaikessa rauhassa loppuun edellisen kierroksensa, jolloin keskeytyskohdasta ei enää olekaan mitään tietoa.

NMI-keskeytykset

Jos osaa käyttää IRQ-keskeytyksiä, osaa myös NMI-keskeytysten salat, sillä toimintaperiaatteissa ei ole eroa. Taulukossa 3.1 luetellut rekisterit ja niiden bitit ovat identtisiä, osoitteisiin on ainoastaan lisättävä $0100.

Normaalisti NMI-keskeytyksiä ei ole kytketty päälle. Restore-näppäin on kuitenkin siitä erikoislaatuinen, että se on kytketty suoraan NMI:n keskeytyslinjaan, toisin sanoen sen hakkaaminen aiheuttaa aina NMI-keskeytyksen. Niinpä onkin syytä määritellä NMI-vektori osoitteisiin $FFFA ja $FFFB, jos Kernal kytketään pois päältä. Muutoin Restore-näppäin voi aiheuttaa koneen sekoamisen. Varsinaisessa NMI-rutiinissa ei tarvitse olla muuta kuin RTI-käsky.

Jos haluaa käyttää NMI:tä IRQ:n tapaan, Restoren aiheuttamasta keskeytyksestä ei ole vakavampaa haittaa, se ei esimerkiksi häiritse ajastimien toimintaa. Halutessaan asiattoman keskeytyksen voi tunnistaa siitä, että kontrollirekisterin $DD0D arvona on nolla, toisin sanoen keskeytyslähde on määrittelemätön.

Toinen merkittävä ero IRQ-keskeytyksiin on siinä, että SEI-käsky ei estä NMI:tä. Tästä on hyötyä sellaisessa tilanteessa, että keskeytyksissä runsaasti erityyppisiä rutiineita, joista jotkut pitäisi välillä kytkeä pois, toisten taas pitäisi pyöriä jatkuvasti. Hyvä esimerkki on pelin taustamusiikki, jota voi soitella NMI-keskeytyksillä, jolloin muilla keskeytyksillä saa pelleillä mielin määrin ilman pelkoa musiikin katkeilemisesta tai rytmin sekoamisesta.

NMI:n kiintoisin ominaisuus on kyky keskeyttää IRQ-keskeytys, kun taas IRQ ei voi keskeyttää NMI:tä. IRQ siis aina odottaa kiltisti vuoroaan, mutta NMI-rutiiniin hypätään heti. Paras esimerkki tämän ominaisuuden soveltamisesta on digitoidun musiikin soittaminen NMI:llä. Ohjelmoija voi olla varma, että samplet käsitellään mahdollisimman tasaisin välein, tapahtui ohjelmassa mitä tahansa.

Vektorien käytöstä

Hyppyvektoreissa on yksi kiusallinen piirre, joka on hyvä muistaa. Nimittäin vektoria ei voi sijoittaa osoitteeseen, jonka alempi tavu on $FF. Tämä johtuu siitä, että prosessorin lukiessa vektoria käskyllä JMP (VEKTORI), se jostain syystä olettaa, että osoitteen ylempi tavu ei muutu. Jos esimerkiksi vektori on sijoitettu osoitteisiin $C0FF ja $C100, prosessori lukeekin vektorin osoitteista $C0FF ja $C000. Olisi pieni ihme, jos ohjelma tällaisessa tilanteessa toimisi – todennäköisyys n. 0.004!

Muistipaikat	Hyppyvektori	Tyyppi
$FFFA, $FFFB	$FE43	NMI
$FFFC, $FFFD	$FCE2	RESET
$FFFE, $FFFF	$FF48	IRQ

Hardware-vektorien sijainti muistissa ja niiden sisältämät hyppyosoitteet. RESET-vektorin muuttamisesta ei ole hyötyä, sillä resetin yhteydessä Kernal kytkeytyy automaattisesti takaisin päälle.

Muistipaikat	Hyppyvektori	Tyyppi
$0314, $0315	$EA31	IRQ
$0316, $0317	$FE66	BREAK
$0318, $0319	$FE47	NMI

Software-vektorien sijainti muistissa ja hyppyosoitteet.

Videopiirin rasterikeskeytykset

Peliohjelmoija ei välttämättä tarvitse lainkaan tavallisia IRQ-keskeytyksiä, sillä videopiirin keskeytykset eli rasterikeskeytykset ovat elegantein tapa tahdistaa ohjelman toiminta kuvaruudun tapahtumiin. Erityisesti grafiikan liikuttelussa on välttämättä tiedettävä missä kohdassa kuvaruudun päivitys on menossa, jotta vältettäisiin epämiellyttävä vilkkuminen ja värinä.

Rasterin rakenne

Kuvan päivitystaajuus on 1/50 sekuntia, toisin sanoen elektroninen suihku piirtää kuvan uudelleen 50 kertaa sekunnissa. Taajuus on niin suuri, että silmä ei erota kuvan vilkkumista. Piirtäminen aloitetaan kuvan yläreunasta ja jokainen rasteririvi piirretään vasemmalta oikealle.

Kuusnelosen ruudulla on kaikkiaan 312 rasteririviä, joista näkyvän alueen osuus on 200 riviä, loput ovat ylä- ja alareunusta.

Rasterivertailu

Muistipaikassa $D012 on rasterirekisteri, jonka sisältö kertoo millä rivillä kuvaruudun päivitys on menossa. Yksi tavu ei kuitenkaan riitä, vaan tarvitaan vielä yhden bitin verran informaatiota, sillä rivi voi osua välille 256–312. Rasterirekisterin ylivuotobitti on muistipaikan $D011 ylin bitti eli bitti 7. Tämä on sikäli käytännöllistä, että muistipaikan lukemisen jälkeen voi käyttää ehdollisia hyppykäskyjä BMI ja BPL, joilla tutkitaan luvun etumerkkiä.

Kuvaruudulla näkyvät rasteririvit osuvat välille $33–$FA (51–250). Oikeastaan ruudulla näkyvät rivit $32–$F9, mutta rasterivertailu ottaa automaattisesti yhden rivin ennakkoa. Tähän hämmennystä aiheuttavaan ratkaisuun on täysin järkevä selitys. Syynä ovat tietyt ajoitusongelmat, joihin palataan tarkemmin aivan luvun lopussa.

Rasterirekistereitä tutkimalla voi muuttaa kuvan sisältöä tietyssä kohdassa kuvaruutua. Seuraavassa esimerkissä vaihdetaan kehyksen väriä aina rasteririvillä nolla, jolloin tuloksena on siististi ilman saumakohtia vilkkuva kehys:

```
        SEI
LOOP    LDA $D012
        BNE LOOP
        LDA $D011    ;varmistetaan ettei
                     ;olla alueella 256-312
        BMI LOOP
        INC $D020
        JMP LOOP
```

Rasterikeskeytys

Rasterirekisterien jatkuva lukeminen ja oikean rivin kyttäily ei ole kovin tehokasta. Keskeytysten avulla turhanpäiväisen odottelun voi jättää videopiirin huoleksi.

Keskeytyskohta määritellään kirjoittamalla halutun rivin arvo rasterirekistereihin $D011 ja $D012. Muistipaikalla $D011 on muitakin tehtäviä, joten ainoastaan bittiä 7 saa mennä näpelöimään. Huomasit varmaankin pienen omituisuuden rasterirekisterien toiminnassa: niihin kirjoitetaan vertailuarvo, mutta niistä luetaan rasteririvi, jolla sillä hetkellä ollaan.

Rasterivertailuun perustuvat keskeytykset käynnistetään kirjoittamalla ykkönen muistipaikkaan $D01A. Nyt prosessori saa aina automaattisesti keskeytyspyynnön, kun kuvaruudun päivitys saapuu ohjelmoijan määrittelemälle rasteririville. Aivan kuten IRQ-keskeytyksissä, myös rasterivertailussa keskeytys on kuitattava. Oikeaoppinen tapa olisi tutkia mikä bitti on päällä tilarekisterissä $D019 ja kirjoittaa sinne sama bitti. Tämä siksi, että videopiirissä on muitakin keskeytyslähteitä kuin rasterivertailu. Muita rasterikeskeytyksiä käytetään kuitenkin sen verran harvoin, että yleensä tarvitsee vain lukea rekisterin arvo ja kirjoittaa se sinne heti takaisin.

Seuraavissa rutiininpätkissä määritellään keskeytyskohdaksi $FB (251) ja testataan toiminta vilkuttamalla kehystä:

```
RASTER SEI
       LDA #$7F        ;IRQ-keskeytykset pois
       STA $DC0D
       LDA #$35        ;Kernal pois
       STA $01
       LDA #<RASCMP    ;keskeytysvektorin alatavu
       STA $FFFE
       LDA #>RASCMP    ;keskeytysvektorin ylätavu
       STA $FFFF
       LDA #$01        ;rasterivertailu päälle
       STA $D01A
       LDA $D011       ;rasterialueeksi 0-255
       AND #$7F        ;ei kosketa bitteihin 0-6
       STA $D011
       LDA #$FB        ;rasteririviksi $FB (251)
       STA $D012
       LDA $DC0D       ;varmistetaan ettei päällä ole
                       ;IRQ-pyyntö
       CLI
WAIT   JMP WAIT        ;tai RTS, ihan miten vain
RASCMP STA $FB
       STX $FC
       STY $FD
       LDA $D019       ;kuitataan rasterikeskeytys
       STA $D019
       INC $D020       ;vaihdetaan kehyksen väri
       LDA $FB
       LDX $FC
       LDY $FD
       RTI
```

Alustusrutiinissa oleva IRQ-keskeytyksen kuittaus käskyllä LDA $DC0D on välttämätön, sillä huonolla onnella rutiinin keskelle saattaa osua keskeytyspyyntö. Sen seurauksena ohjelman suoritus siirtyisi välittömästi CLI-käskyn jäl-

keen keskeytysrutiiniin eikä palaisi sieltä koskaan takaisin, sillä prosessorille tarjotaan "ikuista IRQ-pyyntöä".

Huomaa, että RASCMP-rutiinissa tallennetaan myös X- ja Y-rekisterit, vaikka niitä ei edes käytetä. Tämä on hyvä ottaa tavaksi, sillä kun rutiinia myöhemmin alkaa laajentaa, tällaiset pikku yksityiskohdat taatusti unohtuvat ja seuraava viikko kuluu rattoisasti bugien metsästyksessä.

Keskeytysten vuorottelu

Tietyissä tilanteissa yhden kuvaruutupäivityksen aikana tarvitaan useita rasterikeskeytyksiä eri kohdissa. Kuvitellaan esimerkiksi kahden pelattavaa ammuskelupeliä, jossa kummallakin pelaajalla on oma ruutu. Tällöin kuva pitäisi jakaa neljään osaan: ensimmäisen pelaajan pistetaulu, vieritettävä alue, kakkospelaajan pistetaulu ja toinen vieritettävä alue.

Keskeytyksen voi määrätä vain yhteen kohtaan kerrallaan. Jotta kuvan jakaminen olisi mahdollista, jokaisessa keskeytyksessä on muutettava keskeytyskohta ja hyppyvektori seuraavaa rutiinia varten. Viimeisessä keskeytyksessä luonnollisesti alustetaan ensimmäinen, jotta keskeytykset linkittyisivät eräänlaiseksi päättymättömäksi silmukaksi.

Menetelmällä saa aikaan monia hienoja efektejä, esimerkiksi ruudulla voi olla enemmän kuin kahdeksan spritea ja vierityksen nopeutta voi muuttaa useassa kohdassa, jolloin tuloksena on ns. parallaksiefekti.

Jos keskeytyksiä tarvitaan ainoastaan riveillä 0–255, muistipaikan $D011 ylin bitti on nollattava vain kerran, sillä videopiiri kyllä muistaa mikä arvo sinne on viimeksi kirjoitettu.

Harvinaisemmat keskeytystyypit

Rasterivertailun lisäksi keskeytyslähteenä voi olla myös spriten törmäykset ja valokynä.

Törmäyksiin perustuvat keskeytykset toimivat siten, että keskeytyskohtana on rasteririvi, jolla törmäys ensimmäisen kerran havaittiin. Törmäyksiä on kahta tyyppiä ja kummallekin löytyy oma kontrollibitti muistipaikasta $D01A: bitti 1 spriten ja taustan törmäykselle ja bitti 2 kahden spriten väliselle törmäykselle. Keskeytys on kuitattava tavalliseen tapaan, jotta keskeytyspyyntö ei jäisi päälle. Uutta keskeytystä ei tule ennen kuin on luettu törmäysrekisteri, $D01E spriteille ja $D01F taustalle. Keskeytykset eivät myöskään käynnisty, jos törmäysrekistereissä on nollasta poikkeava arvo.

Spritekeskeytyksistä ei ole juuri mitään käytännön hyötyä. Ensinnäkin yhden kuvaruutupäivityksen aikana on hyvin hankalaa tutkia useampi kuin yksi törmäys ja toiseksi useamman kuin kahden spriten välisiä törmäyksiä on joka tapauksessa mahdotonta päätellä pelkästään törmäysrekistereistä. Aiheeseen palataan tarkemmin spriteja käsittelevässä luvussa.

Valokynälle on peliohjelmoinnissa vähän käyttöä, itse asiassa en ole koskaan onnistunut edes näkemään moista vempainta. Sen sijaan kuusneloseen on mahdollista hankkia valopistooli, joka toimii samalla periaatteella. Valopistoolin keskeytyksiä kontrolloi muistipaikan $D01A bitti 3. Keskeytyksen aiheuttajana on pelaajan liipasinsormi, mikäpä muukaan. Tähtäyskohdan vaaka- ja pystykoordinaatit voi lukea muistipaikoista $D013 ja $D014. Vaakakoordinaatissa on vain kahden pikselin tarkkuus, jotta luku mahtuisi yhteen tavuun. Rekisterissä $D013 on siis vaakakoordinaatti jaettuna kahdella.

Rasterin ajoitus

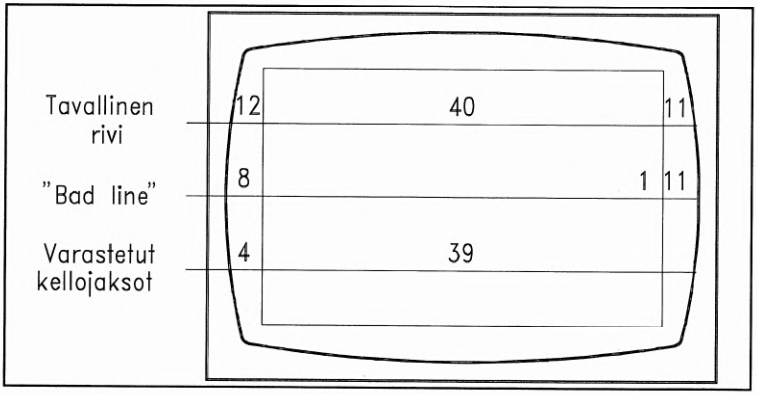
Videopiirin rasterikellon taajuus on noin 8 MHz, mikä käytännössä tarkoittaa sitä, että kahdeksan pikselin piirtämiseen kuluu aikaa tasan yksi kellojakso. Saman voisi ilmaista niinkin, että jokaisen kellojakson aikana videopiiri pystyy siirtämään ruudulle yhden tavun verran tietoa.

Jokaista kuvaruudun näkyvän alueen vaakariviä kohden videopiiri käyttää 63 kellojaksoa. Tästä 40 jaksoa kuluu ruudun alueella, onhan kuvan leveys juuri 40 merkkiä. Loput 23 jaksoa tarvitaan sivureunusten piirtämiseen.

Normaalisti videopiiri ja prosessori käyttävät omia kellojakson puolikkaita, joten ne voivat toimia sulassa sovussa häiritsemättä toisiaan. Joka kahdeksas rasteririvi on kuitenkin poikkeus: videopiirin on “lainattava” kellojaksoja prosessorilta, jotta se ehtisi päivittää sisäisiä grafiikkaosoittimiaan. Näistä rasteririveistä käytetään usein nimitystä "bad line" eli "pahat linjat". Niiden aikana prosessorilta varastetaan yhteensä 43 kellojaksoa. Näistä 40 käytetään osoittimien päivitykseen ja loput kolme ovat eräänlaista ylimenoaikaa, jolla varmistetaan, että prosessori ehtii luovuttaa väylän videopiirille.

Normaalilla rasteririvillä ajankäyttö jakaantuu siten, että vasemman reunuksen kohdalla on aikaa 12 jaksoa, kuvan aikana 40 jaksoa ja sen jälkeen 11 jaksoa. “Bad linella” puolestaan vastaava jakauma on vasemmalla 8 jaksoa, aivan kuvan oikeassa reunassa 1 jakso ja oikealla 11 jaksoa. Prosessorilta varastetaan yhteensä neljä jaksoa ennen kuvaruutua ja videopiiri saa työnsä valmiiksi yhtä sykliä ennen oikeanpuoleista reunusta (kaavio 4.1).

Spritet aiheuttavat ylimääräisiä ajoitusongelmia, sillä myös niiden päivityksessä videopiiri tarvitsee ylimääräisiä kellojaksoja. Yksi sprite varastaa pro-



Kaavio 4.1. Prosessorin käyttöön jäävät kellojaksot tavallisella rasteririvillä ja "bad linella" sekä prosessorilta varastettujen jaksojen sijainti.

sessorilta kaksi kellojaksoa jokaista peittämäänsä rasteririviä kohden. Lisäksi on otettava huomioon, että myös sprite-datan päivitys vaatii ylimääräisen kolmen kellojakson mittaisen viiveen. Kadonneiden kellojaksojen määrä saadaan siis kaavasta:

```
KELLOJAKSOT = 21 X (SPRITET X 2 + 3)
```

Esimerkiksi kahdeksan spritea kuluttaa 19 kellojaksoa riviä kohden, eli yhteensä 399 kellojaksoa.

Videopiirin aloittaa spritedatan lukemisen kuvassa oikeassa laidassa ja tarpeen vaatiessa sitä jatketaan seuraavalla rivillä. Jos kaikki kahdeksan spritea osuvat samalle riville, edellisen rivin oikeassa reunassa käytetään 12 kellojaksoa ja seuraavan rivin vasemmassa reunassa 7 jaksoa.

Hienosäätöä

Yleensä rasterikeskeytysten tarkoituksena on muuttaa vieritysrekistereitä, taustaväriä tai muuta kuvaruudun sisältöä tietyllä rasteririvillä. Nämä vaihdokset olisi hyvä saada reunuksen alle piiloon, jotta ruudulla ei näkyisi rumaa vilkkumista.

Ajoitusta hieman hankaloittaa se, että keskeytyspyynnön ja keskeytykseen hyppäämisen välillä oleva viive hieman vaihtelee. Tämä johtuu siitä, että prosessori ei voi jättää kesken konekielisen käskyn suorittamista, niinpä se huomaakin keskeytyspyynnön vain käskyjen välissä. Kuusnelosessa käskyjen suo-

$D011	Bitti 7: rasterirekisterin bitti 8
$D012	Rasterirekisteri
	Lue rasteririvi/ kirjoita rasterivertailun rivi
$D019	Rasterikeskeytysten status (1 = keskeytys)
	Bitti 7: kaikki keskeytykset
	Bitti 3: valokynä
	Bitti 2: kahden spriten törmäys
	Bitti 1: spriten ja taustan törmäys
	Bitti 0: rasterivertailu
$D01A	Rasterikeskeytysten kontrollirekisteri (1 = keskeytys päälle)
	Bitit samat kuin rekisterissä $D019

Taulukko 4.1. Rasterikeskeytyksissä tarvittavat rekisterit.

ritusaika vaihtelee ja lisäksi on otettava huomioon, että prosessorilla on pientä puuhaa ennen kuin se ehtii hypätä keskeytysrutiinin alkuun. Aikaa vierähtää yhteensä noin 4–10 kellojaksoa.

Kuvaruudun sisältöä ei mitenkään ehdi muuttaa vasemmanpuoleisen reunuksen alla. Niinpä keskeytyskohta onkin määriteltävä yhtä riviä ylemmäksi, jolloin jää runsaasti aikaa säätää ajoitus niin siistiksi kuin pystyy. Kuusnelosen videopiiristä löytyykin pieni nyanssi, jonka kaikki näkemäni käsikirjat ovat unohtaneet mainita: rasterivertailuun on jo valmiiksi säädetty yhden rivin ennakko. Siis kun rekisterin $D012 mukaan ollaan saavuttu riville $32, itse asiassa ollaankin vasta rivillä $31. Tämä ominaisuus voi joskus aiheuttaa pieniä virheitä spriten sijoittelussa, sillä ne puolestaan käyttävät "oikeita" koordinaatteja.

Taustagrafiikka

Kuusnelosessa grafiikan ylläpidosta huolehtii grafiikkapiiri VIC eli Video Inferface Chip. Kuusnelosen kuten kaikkien muidenkin kotimikrojen grafiikka muodostuu erivärisistä pikseleistä, joilla tarkoitetaan pieniä elektronisesti luotuja kuvapisteitä. Näytön leveys on 320 ja korkeus 200 pikseliä, toisin sanoen kuvan resoluutio on 320 X 200. Värejä on puolestaan käytössä kuusitoista.

Näillä eväillä pitäisi saada aikaan kiinnostava ja elävä tausta pelin tapahtumille. Ensimmäiseksi on päätettävä mitä grafiikkatilaa käyttää, sillä vaihtoehtoja on runsaasti. Monivärinen merkkigrafiikka on peleissä yleisimmin käytettyä. Mutta ennen kuin tutustumme tarkemmin eri grafiikkatiloihin on ymmärrettävä miten videopiiri löytää grafiikan muistista.

Värit

Kun kuusneloseen kytketään virta, sen ruudulla on näkyvissä kolmea väriä: reunukset, kuvaruutu ja grafiikkamerkit. Reunusten värikoodi löytyy rekisteristä $D020 ja taustaväri rekisteristä $D021. Kaikille ruudulla näkyville kirjaimille ja grafiikkamerkkeille voi valita oman värinsä kuudestatoista mahdollisesta.

0	Musta	8	Oranssi
1	Valkoinen	9	Ruskea
2	Punainen	10	Vaaleanpunainen
3	Syaani	11	Tummanharmaa
4	Purppura	12	Keskiharmaa
5	Vihreä	13	Vaaleanvihreä
6	Tummansininen	14	Vaaleansininen
7	Keltainen	15	Vaaleanharmaa

Taulukko 5.1. Kuusnelosen värikoodit.

Merkeille onkin varattu erillinen värimuisti, johon kuuluu tuhat värirekisteriä muistipaikasta $D800 alkaen.

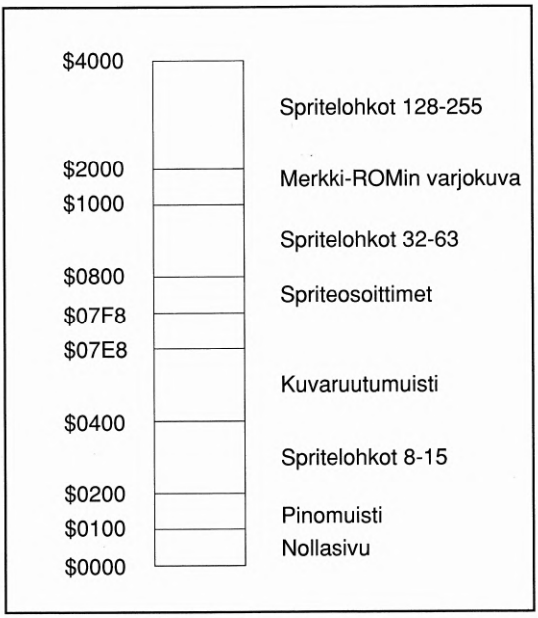
Värimuistin toiminta on siitä erikoista, että jokaisesta tavusta tarvitaan vain neljä alinta bittiä, onhan värejäkin vain kuusitoista. Muut bitit ovat "määrittelemättömiä". Tämä on otettava huomioon värimuistia käsiteltäessä. Sieltä ei välttämättä löydykään sama luku mikä sinne on kirjoitettu, esimerkiksi ykkösen tilalla voikin olla $F1. Värimuistia luettaessa onkin tarpeen maskeerata pois neljä ylintä bittiä.

Videopankkien rakenne

Videopiiri pystyy näkemään kerralla vain 16 kiloa muistia. Kuusnelosen muisti onkin jaettu neljään 16 kilon videopankkiin, joista voi käyttää vain yhtä kerrallaan. Muistipaikan $DD00 kaksi alinta bittiä määräävät mikä pankki on käytössä (taulukko 5.2). Normaalisti videopankki on sijoitettu muistin alkuun alueelle $0000–$3FFF (0–16383).

Kuvaruutumuistin paikka määrätään rekisterin $D018 neljän ylimmän bitin avulla. Merkkigrafiikan sijainti puolestaan määrätään saman rekisterin neljällä alimmalla bitillä (taulukko 5.4). Tosin vain parilliset luvut otetaan huomioon, siis ensimmäisellä bitillä ei ole mitään merkitystä. Normaalitilassa rekisterin arvona on joko $14 tai $15. Taulukosta voidaan lukea, että kuvaruudun pitäisi alkaa osoitteesta $0400 ja merkkigrafiikan osoitteesta $1000. Vaihtamalla pieniin kirjaimiin Commodore- ja Shift-näppäimillä merkkigrafiikka siirtyy alkamaan osoitteesta $1800.

Mutta missä grafiikkamerkit lymyilevät? Ovathan muistialueet $1000– ja $1800– Basicille varattua aluetta, jonne kirjoittaminen ei muuta grafiikkamerkkien muotoa lainkaan. Mystistä. Selitys on siinä, että kuusneloseen valmiiksi muotoilluissa kirjaimissa ja grafiikkamerkeissä on erikoislaatuinen suojaus.



Kuva 5.1. Videopankin sisältö, kun mitään grafiikkarekistereitä ei ole muutettu. Vapaat alueet voi käyttää spriteihin.

Grafiikkamerkit on itse asiassa sijoitettu I/O- piirien alle piiloon. Muistipaikasta $D000 alkavat isot kirjaimet ja muistipaikasta $D800 pienet kirjaimet. Tämä salaperäinen lymypaikka tunnetaan nimellä merkkiROM. Jos merkkigrafiikka osuu välille $1000–$1FFF tai $9000–$9FFF, videopiiri näkeekin merkkiROMin "haamun". Näiden kahden muistialueen erikoisaseman vuoksi niihin ei voi sijoittaa mitään itse tehtyä grafiikkaa, ei edes kuvaruutumuistia.

Valmiita merkkejä ei voi suoraan muuttaa, mutta ne voi lukea merkkiROMista, kun ensin nollataan muistipaikan 1 kolmas bitti. Itse tehtyä grafiikkaa varten ainoastaan toinen ja neljäs videopankki ovat kokonaan ohjelmoijan käytössä. Näistäkin neljäs pankki ($C000-) on hieman epämukava, sillä I/O-piirit ovat tiellä hankaloittamassa ohjelman suunnittelua.

$DD00	Bitit 0-1: videopankin muistialue
	0 = $C000 – $FFFF
	1 = $8000 – $BFFF
	2 = $4000 – $7FFF
	3 = $0000 – $3FFF

Taulukko 5.2. Videopankin määräytyminen.

$D000 – $D7FF	MerkkiROM, isot kirjaimet.
$D800 – $DFFF	MerkkiROM, pienet kirjaimet.
$1000 – $1FFF	MerkkiROMin haamu ensimmäisessä videopankissa.
$9000 – $9FFF	MerkkiROMin haamu kolmannessa videopankissa.

Taulukko 5.3. MerkkiROM ja sen "haamut" muistissa. Haamualueita ei voi käyttää omaan grafiikkaan.

$D018	Bitit 1–3: merkkigrafiikan alkuosoite
	$x0 = $0000 -
	$x2 = $0800 -
	$x4 = $1000 -
	$x6 = $1800 -
	$x8 = $2000 -
	$xA = $2800 -
	$xC = $3000 -
	$xE = $3800 -
	Bitit 4–7: kuvaruudun alkuosoite
	$0x = $0000 -
	$1x = $0400 -
	$2x = $0800 -
	$3x = $0C00 -
	$4x = $1000 -
	$5x = $1400 -
	$6x = $1800 -
	$7x = $1C00 -
	$8x = $2000 -
	$9x = $2400 -
	$Ax = $2800 -
	$Bx = $2C00 -
	$Cx = $3000 -
	$Dx = $3400 -
	$Ex = $3800 -
	$Fx = $3C00 -

Taulukko 5.4. Kuvaruutu- ja merkkigrafiikkamuistin määräytyminen. Kummankin valintaan tarvitaan vain neljä bittiä. Määrittelemätön tavunpuolisko on merkitty pienellä x-kirjaimella. Osoitteisiin on lisättävä videopankin alkuosoite.

Merkkigrafiikka

Normaalitilassa kuusnelosen kuvaruutu koostuu merkeistä, joiden leveys ja korkeus on kahdeksan pikseliä. Näytöllä näkyvät merkit on kuitenkin tallennettu muistiin kuvaruutukoodeina, joita käytetään myös nimitystä poke-koodit. Tästä on käytännön hyötyä, sillä yksi tavu kuvaruutumuistissa vastaa yhtä merkkiä. Kuvaruutu vie muistia vain tuhat tavua ja näytöllä voi olla 256 erilaista grafiikkamerkkiä.

Merkkien muoto on puolestaan tallennettu merkkimuistiin. Normaalisti se alkaa osoitteesta $1000 (4096) ja vie $0800 (2048) tavua. Tämä muistialue on kuitenkin hieman erikoinen. Videopiirille se on ikkuna merkkiROMiin, joka on piilossa I/O-piirien alla. Näin ollen merkkien muotoa ei voi muuttaa ennen kuin merkkimuisti on siirretty toiseen paikkaan. Operaation voi tehdä esimerkiksi seuraavasti:

```
LDA #$18
STA $D018
```

Grafiikkarekisteriin kirjoitetun luvun ylätavu $1 määrää kuvaruudun alkuosoitteeksi $0400, mikä on sama kuin koneen alkuasetus. Alatavuun kirjoitettu $8 vaihtaa merkkimuistin alkamaan ositteesta $2000.

Merkkien muoto muodostuu siten, että jokainen tavu vastaa yhtä pikseliriviä vaakasuunnassa. Bitit puolestaan tulkitaan samoin kuin binaariluvuissa normaalisti, siis ylin bitti on vasemmalla, alin oikealla. Tutkitaan esimerkiksi A-kirjaimen muotoa:

1. tavu: 00011000
2. tavu: 00111100
3. tavu: 01100110
4. tavu: 01111110
5. tavu: 01100110
6. tavu: 01100110
7. tavu: 01100110
8. tavu: 00000000

Merkissä nollat ovat taustaväriä (värirekisteri $D021) ja ykkösten väri määräytyy värimuistin mukaan (muistialue $D800–).

Moniväriset merkit

Normaalitilassa merkit ovat tarkkuusgrafiikkaa, jossa yksi bitti vastaa yhtä pikseliä ja merkissä on korkeintaan kahta väriä. Värien määrää yhtä merkkiä kohden voi lisätä movärigrafiikalla (multicolor mode). Se kytketään päälle kirjoittamalla muistipaikan $D016 bitti 4 ykköseksi.

Moniväritilassa merkin bitit käsitellään pareittain. Näin mahdollisten värien määrä kasvaa kahdesta neljään. Samalla kuvan tarkkuus putoaa puoleen, sillä kaksi viereistä pikseliä ovat aina samaa väriä. Taulukossa 5.5 on lueteltu mistä väritiedot löytyvät eri bittipareille.

Havainnollisuuden vuoksi oletetaan että värikoodeiksi on määrätty arvot 0, 1, 2 ja 3. Toisin sanoen taustaväri on musta, rekisterin $D022 määräämä väri valkoinen, rekisterin $D023 väri punainen ja värimuistissa on syaani. Tällöin tavallinen A-kirjain näyttää monivärigrafiikassa tältä:

1. tavu: 00112200
2. tavu: 00333300
3. tavu: 11221122
4. tavu: 11333322
5. tavu: 11221122
6. tavu: 11221122
7. tavu: 11221122
8. tavu: 00000000

Vain muistipaikasta $D800 alkavan värimuistin määräämiä värejä voi vaihdella merkistä toiseen, muut värit ovat yhteisiä kaikille merkeille.

Värimuistiin kirjoitettujen lukujen merkitys poikkeaa normaalista, sillä värejä on käytössä vain kahdeksan. Jos värikoodi on pienempi kuin kahdeksan, merkki on tarkkuusgrafiikkaa. Jos koodi taas on kahdeksan tai suurempi, merkki on moniväritilassa, mutta värit ovat edelleen väliltä 0–7. Sanotaan sama hienommin: moniväritilassa värikoodin bitti 4 määrää onko merkki tarkkuusgrafiikkaa vai monivärigrafiikkaa. Näin on mahdollista sekoittaa kahta värimoodia samaan kuvaan.

Bittipari	Värirekisterit
00	$D021 (taustaväri)
01	$D022
10	$D023
11	Värimuisti ($D800-)

Taulukko 5.5. Grafiikkamerkin bittiparien merkitys monivärigrafiikassa.

Laajennettu taustaväritila

Tarkkuus- ja monivärigrafiikan lisäksi merkeissä voi käyttää erikoislaatuista taustaväritilaa. Se kytketään päälle rekisterin $D011 bitillä 6. Asetuksella ei ole merkitystä, jos on valittu bittikarttagrafiikka. Taustavärimoodia ei myöskään voi käyttää samaan aikaan monivärigrafiikan kanssa.

Laajennetussa taustaväritilassa merkkejä on käytössä 64, siis vain neljännes normaalista. Tämä siksi, että nyt kuvaruutukoodin kaksi ylintä bittiä määräävät mistä luetaan merkin taustavärin koodi (taulukko 5.6). Merkkien määrän rajallisuuden ja kaksivärisyyden vuoksi grafiikkatilasta on vähän käytännön hyötyä.

Kuvaruutukoodit	Koodin bitit 6-7	Taustavärin rekisteri
$00 - $3F	00	$D021
$40 - $7F	01	$D022
$80 - $BF	10	$D023
$C0 - $FF	11	$D024

Taulukko 5.6. Merkin taustavärin määräytyminen laajennetussa taustavärimoodissa.

Bittikarttagrafiikka

Bittikartta eroaa merkkigrafiikasta siinä, että jokaista tavua muistissa vastaa yksi tavu kuvaruudulla. Näin ollen kuvan jokaista pikseliä voidaan käsitellä erikseen. Bittigrafiikka valitaan rekisterin $D011 bitillä 5.

Bittikartalla grafiikka on järjestelty samaan tapaan kuin grafiikkamerkeissä. Tavallaan käytössä on tuhat grafiikkamerkkiä ja kuvaruutu on ikkuna suoraan merkkimuistiin. Oheinen kaavio havainnollistaa bittigrafiikan muodostumista:

Merkki 1	**Merkki 2**	...	**Merkki 40**
1. tavu	9. tavu	...	313. tavu
2. tavu	10. tavu	...	314. tavu
3. tavu	11. tavu	...	315. tavu
4. tavu	12. tavu	...	316. tavu
5. tavu	13. tavu	...	317. tavu
6. tavu	14. tavu	...	318. tavu
7. tavu	15. tavu	...	319. tavu
8. tavu	16. tavu	...	320. tavu
Merkki 41			
321. tavu			
jne.			

Grafiikkamuistin ja kuvaruutumuistin paikka määrätään aivan samalla tavalla kuin merkkigrafiikassakin (taulukko 5.4). Ainoa ero on siinä, että bittikartan koon vuoksi sen alkuosoitteeksi on vain kaksi vaihtoehtoa, nolla tai $2000. Mutta mihin ihmeeseen tarvitaan kuvaruutumuistia käytettäessä bittikarttaa? Sitä tarvitaan siksi, että se mikä merkkigrafiikassa ymmärrettiin kuvaruutumuistiksi onkin bittikartassa osa värimuistia. Bittikartalla värien käsittely siis poikkeaa suuresti merkkigrafiikasta.

Tarkkuusgrafiikkaa käyttävässä bittikartassa normaalilla taustavärirekisterillä $D021 ei ole merkitystä. Sen sijaan värit määräytyvät kuvaruutumuistin tavujen mukaan. Tavun ensimmäiset neljä bittiä määrittelevät värikoodin nollabiteille ja ylemmät neljä bittiä värin ykkösbiteille. Jos kuvaruutumuistissa on esimerkiksi välilyönnin poke-koodi (heksana $20), niin vastaavassa kohdassa bittikartalla tausta on musta (= 0) ja grafiikka punaista (= 2). Aivan kuten merkkigrafiikassakin, värejä voi muuttaa vain yhden merkin kokoisella alueella, ei jokaiselle pikselille erikseen.

Monivärigrafiikassa bittikartan värien muodostuminen on mutkikkaampaa. Taulukko 5.7 havainnollistaa asiaa. Värien muodostumisessa pätee sama logiikka kuin merkkigrafiikassa: bitit käsitellään pareina, jonka seurauksena grafiikan tarkkuus heikkenee. Kuvaruutumuistin sisältö tulkitaan väreiksi samalla periaatteella kuin tarkkuusgrafiikassa, mutta lisäksi on käytössä taustaväri ja ta-

vallinen värimuisti. Merkkigrafiikkaan verrattuna tärkein ero on siinä, että jokaista merkkiä kohden on kolme vapaasti valittavaa väriä. Niinpä lopputuloskin on suunnattomasti värikkäämpi.

Bittikartan bittipari	Värin määräytyminen
00	$D021 (taustaväri)
01	Bitit 4-7 kuvaruutumuistissa
10	Bitit 0-3 kuvaruutumuistissa
11	Värimuisti ($D800-)

Taulukko 5.7. Monivärisen bittikartan värit.

$D011	Bitti 4: 1 = näytä kuva, 0 = sammuta kuva
	Bitti 5: 0 = merkkigrafiikka, 1 = bittikartta
	Bitti 6: 1 = laajennettu taustaväri
$D016	Bitti 4: 0 = tarkkuusgrafiikka, 1 = monivärigrafiikka

Taulukko 5.8. Tarkkuus-, moniväri- ja bittikarttagrafiikan määrittely.

Osoitteen laskeminen bittikartalla

Erikoisen järjestelyn vuoksi oikean kohdan laskeminen bittikartalla on hankalaa. Miten on toimittava, jos halutaan sytyttää yksi pikseli tietyssä kohdassa ruutua? Tämä vaatiikin pientä matemaattista voimistelua. Ensiksi on laskettava merkkirivi ja sarake:

```
RIVI = INT(Y/8)
SARAKE = INT(X/8)
```

Seuraavaksi tutkitaan mitä tavua kyseisen merkin alueella on käsiteltävä:

```
TAVU = Y AND 7
```

Näiden tietojen perusteella voidaankin laskea tavun osoite kuvaruudulla:

```
OSOITE = ALKUOSOITE + RIVI * 320 + SARAKE * 8 + TAVU
```

Vielä on tiedettävä mitä bittiä kyseisessä tavussa on käsiteltävä:

```
BIT = 7 - (X AND 7)
LUKU = 2^BIT
```

Lopuksi laskettuun osoitteeseen kirjoitetaan saatu luku OR-operaatiolla, jotta ei vaikutettaisi tavun muihin bitteihin. Huomaa että bittikarttaruudun osoitetta määriteltäessä sekä X- että Y-koordinaatin laskeminen aloitetaan vasemmasta yläkulmasta.

Edellä kuvattu laskentatapa on havainnollinen, mutta erittäin hidas jopa konekielellä toteutettuna. Niinpä onkin otettava käyttöön pari oikotietä. Ensimmäiseksi kannattaa taulukoida muistiin jokaisen vaakarivin alkuosoite. Muistia tähän kuluu 400 tavua (vaakarivejä on 200), mutta nyt laskeminen käy vaivattomasti. Lisäksi esimerkkirutiinin BITS-taulukkoon on laskettu valmiiksi kahden potenssit ja otettu huomioon bittien käänteinen järjestys. Tämä lisää entisestään rutiinin tehoa:

```
PLOT   LDX YCO      ;Y-koordinaatti X-rekisteriin
       LDA BITLO,X  ;luetaan rivin osoite taulukosta
       STA $FB
       LDA BITHI,X
       STA $FC
       LDA XCO
       TAY          ;X-koordinaatti Y-rekisteriin
       AND #7
       TAX          ;bitin järjestysnumero
                    ;X-rekisteriin
       LDA ($FB),Y  ;luetaan tavu bittikartalta
       ORA BITS,X   ;lisätään taulukosta
                    ;luettava bitti
       STA ($FB),Y  ;kirjoitetaan tavu bittikartalle
       RTS
BITS   .BYTE 128,64,32,16,8,4,2,1
BITLO  .BYTE $00,$01,$02,$03,$04,$05,$06,$07
       .BYTE $40,$41,$42 jne...
BITHI  .BYTE $20,$20,$20,$20,$20,$20,$20,$20
       .BYTE $21,$21,$21 jne...
```

Jos rutiinia haluaa kokeilla käytännössä, on ensin tehtävä valmiiksi taulukot BITLO ja BITHI, jotka sisältävät jokaisen pikselirivin alkuosoitteen. Esimerkissä on malliksi pieni pätkä taulukoista, kun bittikartta alkaa osoitteesta $2000. Joka kahdeksannella rivillä osoitteeseen lisätään $0140 eli 320.

Bittikartan käyttäminen peleissä

Bittikartan käsittely on luonnollisesti merkkigrafiikkaa hitaampaa, onhan grafiikkaa käytössä kahdeksankertainen määrä. Tämä rajoittaakin sen soveltamista peleihin. Parhaiten bittikartta soveltuu alkukuviin ja taustoihin, joissa ei käytetä vieritystä. Tosin vierityskään ei ole mahdoton toteuttaa (kts. luku "Erikoistehosteita videopiirillä"), mutta käytettävä ohjelmointitekniikka on hyvin työläs ja vaikeasti hallittava. Jos on sinnikäs ja pitää haasteista, siihen kannattaa tutustua.

Bittikartan käyttäminen on väistämätöntä silloin, kun ainakin osa kuvasta piirretään ohjelman ajon aikana. Kolmiulotteinen vektorigrafiikka on tästä hyvä esimerkki. Esineen kulmapisteet projisoidaan kaksiulotteiselle pinnalle ja yhdistetään viivoilla. Tuloksena on esineen "rautalankamalli", jota on mahdollista pyörittää ja liikuttaa kolmiulotteisessa avaruudessa. Operaatio vaatii raskaita matemaattisia rutiineita, joihin kuusnelonen ei sovellu erityisen hyvin. Vaikka lopputulos ei tyydyttäisikään, ohjelmoinnista saa joka tapauksessa arvokasta kokemusta.

Bittikarttaa voi käyttää myös isometrisella viistoperspektiivillä toteutettuun grafiikkaan. Viistoperspektiivissä pelikentät ovat yleensä neliömäisiä ja katselupaikka on alueen yhdessä kulmauksessa lattiatason yläpuolella. Grafiikka synnyttää kolmiulotteisen vaikutelman, vaikka siitä puuttuukin aito syvyysvaikutelma.

Viistoperspektiivissä pelihahmojen on voitava liikkua esineiden edestä ja takaa. Kaikki pelikentän kappaleet on piirrettävä ruudulle oikeassa järjestyksessä aina kuvan päivityksen yhteydessä. Tämä vaatii grafiikan yhdistelemistä loogisilla operaatioilla, mihin bittikartta sopii merkkigrafiikkaa paremmin. Aiheeseen palataan luvussa "Liikettä ruudulle".

Taustojen suunnittelu

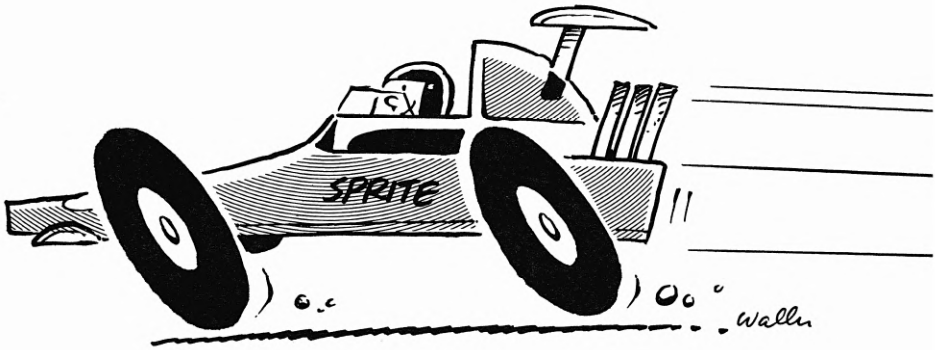
Hyvin tehdyssä taustagrafiikassa on paljon vaihtelua ja merkkien rajakohdat eivät erotu häiritsevästi. Merkkien suunnittelu on kätevintä tehdä ohjelmalla, jolla pystyy tekemään useamman merkin kokoisia hahmoja, olivat ne sitten puita, lehmiä tai lasertykkejä. Apuohjelma voi olla esimerkiksi sellainen, että

puolet kuvasta on varattu ruudukolle, jossa pikselit on kuvattu suurennettuina, loppuosa kuvasta näyttää merkit oikeankokoisina. Paremman puutteessa taustan yksityiskohdat voi tehdä bittikarttaa käyttävällä piirto-ohjelmalla, jonka jälkeen hahmo muutetaan grafiikkamerkeiksi pala kerrallaan.

Myös pelikenttien rakenteluun on järkevintä käyttää apuohjelmaa. Pitkien vierivien taustojen naputteleminen koneeseen merkki kerrallaan on kohtuuttoman vaivalloista. Taustat voi rakentaa merkki kerrallaan tai peruspalana voi käyttää isompaa, esimerkiksi neljän merkin kokoista palaa.

Taustoissa on otettava huomioon myös muistin kulutus. Käytettäessä yhtä merkkiä suurempia peruspaloja taustan voi tallentaa muistiin tiivistetyssä muodossa. Yksi tavu muistissa kertoo mikä paloista piirretään ruudulle, joten mitä isompia peruspaloja käytetään, sitä vähemmän muistia pelin tausta kuluttaa.

Muistia voi säästää myös itse kehiteltyjen koodisysteemien avulla. Yksi mahdollisuus on käyttää taulukoita, joihin tallennetaan ainoastaan grafiikkapalan koodi ja sijaintipaikka ruudulla. Tämä on erityisen tehokas tapa silloin, kun suuri osa taustasta on tyhjänä tai grafiikan alle piirretään ensin toistuva peruskuvio. Ensimmäisestä ovat esimerkkinä monet ammuskelupelit ja jälkimmäisestä Breakoutin tyyppiset pelit, joissa taustana on yksinkertainen peruskuvio ja sen päällä rikottavia tiiliä. Ideaa voi varioida monin tavoin. Pitkän tiiliseinän saa mahtumaan neljään tavuun: Taulukossa on ensimmäisen tiilen paikka pysty- ja vaakasuunnassa sekä seinän leveys ja korkeus. Itse olen erityisen mieltynyt tähän tekniikkaan ja olenkin käyttänyt sitä kaikissa kolmessa julkaistussa pelissäni. Esimerkiksi Netherworldissä sain ahdettuna lähes kolmensadan kuvaruudun kokoisen vierivän alueen kuuteen kiloon muistia.



Spritet

Sprite on itsenäinen graafinen olio, jonka päivittämisestä huolehtii videopiiri. Tavallaan sprite on taustagrafiikan sekaan heijastettu minikokoinen ikkuna. Videopiirissä on erilliset rekisterit kahdeksalle spritelle. Jokaiselle voi määrätä erikseen muodon, värin, grafiikkatilan ja paikan. Näiden ominaisuuksien ansiosta niistä on suunnattomasti apua liikkuvien pelihahmojen ohjelmoinnissa.

Spriten muoto

Spriten ulkonäkö määräytyy sprite-lohkojen sisällön mukaan. Sprite-lohko tarkoittaa muistialuetta, jonka alkuosoite on jaollinen 64:llä. Spritellä on korkeutta kaksikymmentäyksi pikseliä ja yksi pikselirivi vaakasuunnassa muodostuu kolmesta peräkkäisestä tavusta. Yksi kokonainen sprite siis vaatii 63 tavua muistia. Sprite-lohkojen loppuun jää yksi ylimääräinen tavu, josta ei tarvitse välittää.

Yksivärisessä spritessa ykkösbitit ovat spriten omaa väriä. Nollabitit ovat puolestaan läpinäkyviä, toisin sanoen niiden tilalla on taustagrafiikkaa. Spriten muoto kirjoitetaan muistiin vasemmalta oikealle rivi kerrallaan. Kuvassa 6.1 on havainnollistettu miten kuva muodostuu biteistä.

	1. tavu	2.tavu	3. tavu	arvot heksana
1. rivi	00000000	00000000	00000000	$00 $00 $00
2. rivi	00000000	00000000	00000000	$00 $00 $00
3. rivi	00000000	00000000	00000000	$00 $00 $00
4. rivi	00000000	00000000	00000000	$00 $00 $00
5. rivi	00000000	00000000	00000000	$00 $00 $00
6. rivi	00000000	00011000	00000000	$00 $18 $00
7. rivi	00110000	01111110	00001100	$30 $7E $0C
8. rivi	00110111	11111111	11101100	$37 $FF $EC
9. rivi	11111111	11111111	11111111	$FF $FF $FF
10. rivi	00001000	11111111	00010000	$08 $FF $10
11. rivi	00000100	00111100	00100000	$04 $3C $20
12. rivi	00000010	00011000	01000000	$02 $18 $40
13. rivi	00000001	00111100	10000000	$01 $3C $80
14. rivi	00000000	11111111	00000000	$00 $FF $00
15. rivi	00000000	01111110	00000000	$00 $7E $00
16. rivi	00000000	01111110	00000000	$00 $7E $00
17. rivi	00000000	01111110	00000000	$00 $7E $00
18. rivi	00000000	00111100	00000000	$00 $3C $00
19. rivi	00000000	00000000	00000000	$00 $00 $00
20. rivi	00000000	00000000	00000000	$00 $00 $00
21. rivi	00000000	00000000	00000000	$00 $00 $00

Kuva 6.1 Spriten muoto on selvästi nähtävissä, kun sprite-lohkon tavut kirjoittaa binaarilukuina.

Sprite-lohkon valinta

Jokaisessa 16 kilon videopankissa voi periaatteessa olla 256 sprite-lohkoa. Käytännössä niistä voi hyödyntää vain osaa. Esimerkiksi merkkiROMille varattuja alueita ei voi käyttää lainkaan ja nollasivu tai kuvaruutumuisti eivät ole erityisen käytännöllisiä vaihtoehtoja.

Erityisillä osoitinrekistereillä jokaiselle spritelle voi erikseen määrätä mistä lohkosta sen muoto löytyy. Sprite-osoittimet ovat siitä omalaatuisia, että niiden

sijainti muistissa riippuu kuvaruutumuistin sijainnista. Ensimmäisen spriten osoitin löytyy siten, että kuvaruutumuistin alkuosoitteeseen lisätään $03F8 (1016). Normaalisti kuvaruutu alkaa osoitteesta $0400 (1024), joten sprite-osoittimet sijaitsevat osoitteissa $07F8–$07FF (2040–2047).

Miten sitten yhdellä tavulla kuvataan sprite-lohkon muistiosoite, onhan videopankissa 16 kiloa muistia? Sprite-osoittimen arvo lasketaan siten, että sprite-lohkon alkuosoite jaetaan 64:llä. Tulokseksi saadaan sprite-osoitin, jonka arvo on sopivasti välillä 0-255. Mikäli videopankkia on muutettu, sprite-lohkon osoitteesta on aluksi vähennettävä videopankin alkuosoite. Normaalisti se alkaa muistipaikasta nolla.

Sprite-osoittimien ansiosta animaation tekeminen on erittäin helppoa. Ensin on tallennettava muistiin useita versioita samasta sprite-hahmosta siten, että niistä muodostuu kuvasarja, esimerkiksi kauniisti leviävä rajähdyspivi. Tämän jälkeen tarvitsee vain muuttaa sprite-osoitinta sopivalla nopeudella.

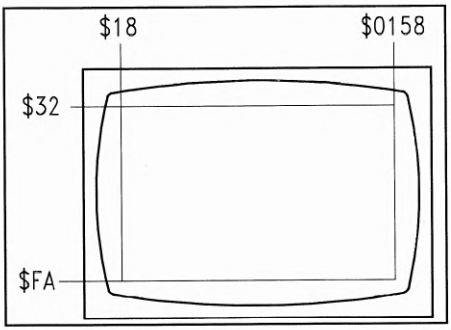
Spriten sijoittaminen näytölle

Videopiiri ei piirrä spriteja ruudulle ennen kuin sitä erikseen käsketään. Rekisterin $D015 avulla spritet voi kytkeä päälle ja pois valikoidusti. Rekisterin kahdeksan bittiä vastaavat kahdeksaa spritea siten, että bitti 0 tarkoittaa ensimmäistä spritea, bitti 1 toista spritea ja niin edelleen. Samaa käytäntöä on sovellettu useisiin muihinkin sprite-rekistereihin. Sprite kytketään päälle kirjoittamalla sitä vastaava bitti ykköseksi. Esimerkiksi jos näytölle halutaan vain neljä ensimmäistä spritea, muistipaikkaan $D015 kirjoitetaan luku $0F, eli binaarina %00001111.

Spriten kytkeminen päälle ei vielä takaa, että se näkyy kuvassa, sillä spriten voi sijoittaa myös reunuksen alle piiloon. Virran kytkemisen jälkeen sekä vaaka- että pystykohdaksi on määritelty nolla, mikä käytännössä tarkoittaa sitä, että ne kaikki lymyilevät piilossa reunuksen alla ylävasemmalla. Sprite on näkyvissä aivan kuvaruudun vasemmassa yläkulmassa, kun sen paikaksi määritellään vaakasuunnassa $18 ja pystysuunnassa $32.

Spriten sijainti kuvaruudulla määritellään rekistereillä $D000-$D010. Ensimmäisen spriten X-koordinaatti on muistipaikassa $D000 ja sen) Y-koordinaatti muistipaikassa $D001. Seuraavissa kahdessa muistipaikassa on toisen spriten X- ja Y-koordinaatti, ja niin edelleen.

Yksi tavu ei aina riitä X-koordinaatin määrittelyyn, sillä kuvaruudulla on leveyttä 320 pikseliä. Tarvitaan vielä yhden bitin verran informaatiota, jotta voitaiisin kuvata koordinaatit 256-319. Tähän tarkoitukseen käytetään muistipaikkaa $D010, jonka bitit vastaavat kahdeksaa spritea tavalliseen tapaan. Siis



Kuva 6.1 Spriten vaaka- ja pystykoordinaatit kuvaruudun rajakohdissa, kun kuvaa ei ole kavennettu eikä sprite- ja laajennettu.

bitti 0 vastaa ensimmäistä spritea, bitti 1 toista spritea, jne.

Kokeillaan esimerkiksi siirtää ensimmäinen sprite aivan kuvan oikeaan reunaan. Sen paikaksi vaakasuunnassa on määrättävä $0140. Rekisterin $D010 bitti nolla kirjoitetaan ykköseksi ja muistipaikkaan $D000 kirjoitetaan $40.

Spriten värit

Spriteille on valittavana kaksi grafiikkatilaa, yksi- ja monivärinen. Muistipaikka $D01C määrää kumpi on käytössä. Kun spritea vastaava bitti kirjoitetaan ykköseksi, sprite on moniväritilassa.

Jokaiselle spritelle on oma värirekisteri muistipaikoissa $D027–$D02E. Yksiväritilassa ainoastaan näiden rekisterien arvoilla on merkitystä. Moniväritilassa jokaisessa spritessa on kolme väriä sekä läpinäkyvä "väri". Väreistä kaksi on kaikille spriteille yhteisiä ja ne valitaan kirjoittamalla värikoodi rekistereihin $D025 ja $D026. Moniväritilassa spriten pikselit järjestellään pareiksi aivan kuten muussakin grafiikassa. Bittiparin arvo määrää mitä värirekisteriä käytetään.

Bittipari	Värirekisteri
00	läpinäkyvä (kuvaruudun väri)
01	$D025
10	Spriten oma väri, rekisterit $D027-$D02E
11	$D026

Taulukko 6.1 Monivärispriten värien määräytyminen. Vain yksi kolmesta väristä voidaan valita erikseen jokaiselle spritelle.

Spriten prioriteetit

Prioriteetilla tarkoitetaan kirjaimellisesti "tärkeysjärjestystä". Videopiirin on tiedettävä missä järjestyksessä spritet piirretään ruudulle silloin kun kaksi tai useampia spriteja osuu päällekkäin. Kahden spriten keskinäistä prioriteetia ei voi muuttaa. Ne piirretään aina siten, että pieninumeroisin sprite peittää alleen kaikki muut. Kahdeksas sprite puolestaan jää kaikkien muiden alle. Jos prioriteettien vaihtamista tarvitaan pelin aikana, se on toteutettava ohjelmallisesti. Käytännössä se tarkoittaa sitä, että vaihdetaan keskenään kahden spriten kaikkien rekisterien arvot.

Toinen spriteihin liittyvä prioriteetti on spriten ja taustagrafiikan järjestys. Muistipaikan $D01B avulla prioriteetin voi määrätä erikseen kaikille spriteille. Jos spritea vastaava bitti kirjoitetaan ykköseksi, taustagrafiikka piirretään ennen spritea. Tällöin sprite-hahmo näkyy vain sellaisissa kohdissa, joissa on taustaväriä.

Jos taustagrafiikka on moniväritilassa, asia mutkistuu hieman. Monivärisessä taustagrafiikassa värit määräytyvät tavalliseen tapaan neljän erilaisen bittiparin mukaan, mutta näistä vain bittiparit 10 ja 11 peittävät spriten alleen (taulukko 6.2). Ominaisuuden ansiosta spritea voi liikutella osaksi grafiikan alla ja osaksi sen päällä. Hyvällä grafiikan suunnitellulla saa aikaan esimerkiksi hienon kolmiulotteisen vaikutelman. Toinen mahdollisuus on aidon näköisen varjon tekeminen pelihahmolle. Tällöin bittiparit 11 ja 10 käytetään taustaväriin sekä grafiikan tummimpaan väriin, joka on sama kuin varjon väri. Varjo siis peittää alleen grafiikan kaksi vaaleinta sävyä, joissa on käytetty bittipareja 00 ja 01.

Taustan bittipari	Näkyvyys
00	sprite näkyvissä
01	sprite näkyvissä
10	sprite grafiikan alla
11	sprite grafiikan alla

Taulukko 6.2 Spriten näkyvyys silloin, kun tausta on moniväritilassa ja sille on määritelty spritea suurempi prioriteetti rekisterin $D01B avulla.

Spriten laajentaminen

Spriten koon voi kaksinkertaistaa sekä vaaka- että pystysuunnassa. Rekisterillä $D017 määrätään koko pystysuunnassa ja rekisterillä $D01D koko vaakasuunnassa. Jos spritca vastaava bitti on ykkönen, sprite on laajennettu.

Laajennettuja spriteja on käytettävä harkiten, sillä kaksinkertaiseksi venytettyjen pikselit tekevät sprite-hahmosta karkean palikkamaisen. Eniten käyttöä laajennukselle on silloin, kun tarvitaan koko ruudun mittaisia palkkeja vaaka- tai pystysuunnassa. Esimerkiksi jaettaessa ruutu kahteen osaan pystyvierityksessä ruudulla on sottainen vaihtumiskohta, mutta sen saa kätevästi piiloon vaakasuunnassa laajennetuilla spriteilla.

Törmäysten tutkiminen

Spriteillä on kaksi törmäysrekisteriä. Rekisteristä $D01E voi tutkia sprite-törmäykset ja rekisteristä $D01F spriten ja taustagrafiikan väliset törmäykset. Rekisterien bitit vastaavat spriteja tavalliseen tapaan. Rekistereitä luettaessa niiden arvot nollataan automaattisesti.

Kun törmäyksessä on mukana enemmän kuin kaksi spritca, rekisteristä ei ole erityisemmin hyötyä. Esimerkiksi jos rekisterin $D01E arvona on $47 (%01000111 binaarina), on mahdoton tietää onko ensimmäinen sprite törmännyt toiseen, kolmanteen vai seitsemänteen spriteen. Niinpä sprite-törmäykset on selvitettävä tutkimalla kahden spriten välinen etäisyys.

Spriten ja taustan välisissä törmäyksissä rekisteristä $D01F saa luotettavampaa tietoa. Tosin tällöinkin on muistettava yksi erikoistilanne: monivärifiikalla tehdyn taustan bittipari 01 ei aiheuta törmäystä. Arvatenkin syntyy ongelmia, jos suuri osa taustasta on piirretty kyseisen bittiparin määrittelemällä värillä. Merkkigrafiikalla tehdyissä taustoissa törmäykset saa luotettavimmin selville tutkimalla spriten alla olevia grafiikkamerkkejä, mutta tällöin törmäysten tarkkuus hieman kärsii.

Spriten monistaminen

Ohjelmoijan ei tarvitse tyytyä kahdeksaan spriteen. Samaa spritea voi nimittäin piirtää ruudulle useaan kertaan. Operaatio vaatii ainostaan tarkkaa ajoitusta. Kun spriten paikka ja muut ominaisuudet on määritelty kertaalleen, odotellaan juuri spriten alla olevaa rasteririviä. Sitten siirretään spritea alemmaksi. Samalla voi muuttaa spriten muotoa, väriä, X-koordinaattia ynnä muita ominai-

suuksia. Yhdestä spritesta voi tehdä niin paljon kopiota kuin ruudulle pystysuunnassa mahtuu, käytännössä yhdeksän kappaletta.

Lähes kaikkia spriten ominaisuuksia voi muuttaa myös spriten piirtämisen aikana. Ainoa poikkeus on spriten paikka pystysuunnassa. Y-koordinaatin muuttamisella ei ole vaikutusta ennen kuin sprite on piirretty kokonaan, toisin sanoen spritea ei voi katkaista keskeltä. Spriten kopio ei myöskään näy ruudulla mikäli se on päällekkäin edellisen kanssa.

Spritea ei voi monistaa vaakasuunnassa, sillä X-koordinaatin vaihtaminen ainoastaan halkaisee spriten. Spriteja voi siis olla ruudulla vierekkäin vain kahdeksan. Ainoa tapa saadaa vaakariville enemmän spriteja on näyttää niitä vuorotellen eri kohdissa spriten pysyessä yhdessä paikassa 1/25 sekuntia kerrallaan.

Sprite-hahmojen monistus on nopeutta vaativa operaatio. Niinpä sprite-rekistereitä ei kannata käsitellä suoraan, vaan on otettava käyttöön "haamurekistereitä". Niillä tarkoitetaan yksinkertaisesti itse määriteltyjä rekistereitä, joissa on kaikki spriteihin liittyvät tiedot. Esmerkiksi animaatio ja spriten liikutus on mahdollista laskea valmiiksi kaikille spriteille, myös niiden klooneille, ja kuvan piirron aikana tiedot ainoastaan kopioidaan oikealla hetkellä varsinaisiin sprite-rekistereihin.

Taulukko 6.3. Sprite-rekisterit. Rekistereissä välillä $D010–$D01F jokaiselle spritelle on oma bittinsä.

$D000	1. sprite, X-koordinaatti
$D001	1. sprite, Y-koordinaatti
$D002	2. sprite, X-koordinaatti
$D003	2. sprite, Y-koordinaatti
$D004	3. sprite, X-koordinaatti
$D005	3. sprite, Y-koordinaatti
$D006	4. sprite, X-koordinaatti
$D007	4. sprite, Y-koordinaatti
$D008	5. sprite, X-koordinaatti
$D009	5. sprite, Y-koordinaatti
$D00A	6. sprite, X-koordinaatti
$D00B	6. sprite, Y-koordinaatti
$D00C	7. sprite, X-koordinaatti
$D00D	7. sprite, Y-koordinaatti

$D00E	8. sprite, X-koordinaatti
$D00F	8. sprite, Y-koordinaatti
$D010	0 = X-koordinaatti 0-255
	1 = X-koordinaatti yli 255
$D015	0 = sprite pois päältä
	1 = sprite päällä
$D017	0 = sprite normaalikokoinen pystysuunnassa
	1 = sprite laajennettu pystysuunnassa
$D01B	0 = sprite taustagrafiikan päällä
	1 = sprite taustagrafiikan alla
$D01C	0 = yksivärinen sprite
	1 = monivärinen sprite
$D01D	0 = sprite normaalikokoinen vaakasuunnassa
	1 = sprite laajennettu vaakasuunnassa
$D01E	1 = sprite törmännyt toiseen spriteen
$D01F	1 = sprite törmännyt taustaan
$D025	1. Monivärirekisteri
$D026	2. Monivärirekisteri
$D027	1. spriten väri
$D028	2. spriten väri
$D029	3. spriten väri
$D02A	4. spriten väri
$D02B	5. spriten väri
$D02C	6. spriten väri
$D02D	7. spriten väri
$D02E	8. spriten väri

Liikettä ruudulle

Liikkuvat ja animoidut hahmot ovat toimintapelien sielu. Liikkeiden on oltava mahdollisimman sulavia ja luontevia, jotta pelaaminen olisi nautittavaa. Pelihahmoissa on kätevintä käyttää spriteja, mutta myös merkkigrafiikalla saa aikaan siistiä jälkeä, tosin sen ohjelmoiminen on työläämpää. Sprite-grafiikan rajoitukset pakottavat usein käyttämään rinnakkain sekä spriteja että liikkuvaa merkkigrafiikkaa.

Liikkuvat spritet

Jotta spriten liike näyttäisi täysin pehmeältä, sitä on siirrettävä kerran jokaisen kuvaruutupäivityksen aikana. Näin ollen liikkeen nopeus ei suinkaan tarkoita, miten nopeasti spritea siirretään, vaan miten monta pikseliä kerrallaan sitä liikutetaan. Hitain "pehmeä" nopeus on 50 pikseliä sekunnissa ja ylärajaa ei ole. Tosin käytännössä kovin suuria nopeuksia ei kannata käyttää.

Mikäli nopeuden säätäminen pikselin tarkkuudella tuntuu liian karkealta, voi ottaa käyttöön desimaaliluvut. Yksinkertaisin tapa on kuvata sekä nopeus että spriten koordinaatit kahdella tavulla, jolloin alempi tavu tulkitaan desimaaliosaksi. Spriten paikka määräytyy ylemmän tavun mukaan, kun taas desimaaliosa tarkoittaa "pikselin murto-osia":

```
MOVESP CLC
       LDA XDESIM  ; X-koordinaatin desimaaliosa
       ADC SDESIM  ; lisätään nopeuden desimaaliosa
       STA XDESIM
       LDA XKOKON  ; X-koordinaatin kokonaisosa
       ADC SKOKON  ; lisätään nopeuden kokonaisosa
       STA XKOKON
       STA $D000   ; luku kordinaattirekisteriin
       RTS
```

Esimerkissä on yksinkertaisuuden vuoksi oletettu, että spritea liikutetaan ainoastaan alueella 0–255. Nopeudessa voidaan käyttää myös negatiivisia lukuja (kokonaisosa = $80–$FF).

Käyttämällä kahta tavua pienin mahdollinen nopeus on 1/255 pikseliä kuvan

päivitystä kohden (noin 0.2 pikseliä sekunnissa). Yleensä pienempikin tarkkuus riittää. Niinpä desimaaliosassa tarvitsee käyttää vain muutamaa ylintä bittiä.

Tarkasti säädettävästä nopeudesta on eniten hyötyä silloin, kun halutaan muuttaa liikkeen suuntaa mahdollisimman pehmeästi. Varsinkin pelihahmon ohjaaminen tuntuu huomattavasti miellyttävämmältä, jos se ei siirtyile töksähdellen, vaan sen liikkeissä on hitunen inertiaa eli hitausvoimaa. Alussa on lyhyt kiihdytys ennen kuin hahmo liikkuu täydellä nopeudella. Inertiaa ei missään tapauksessa saa liioitella, sillä tällöin ohjaus muuttuu kankeaksi.

Spriten liikeradat saadaan aikaan muuttamalla sen nopeutta yksinkertaisen kaavan tai taulukon mukaan. Taulukkoon voi sijoittaa joko absoluuttiset nopeudet tai nopeuden muutokset. Tietysti on mahdollista taulukoida valmiiksi kaikki spriten paikat ruudulla, mutta tämä ei ole erityisen suositeltava tapa suuren muistinkulutuksen ja systeemin kankeuden vuoksi.

Liikkuvat merkit

Merkkigrafiikalla tehtyjen pelihahmojen liikuttaminen on huomattavasti spriteja työläämpää, sillä niiden paikan voi määrätä vain kahdeksan pikselin tarkkuudella. Yleensä merkkejä käytetäänkin pienissä ja nopeasti liikkuvissa kappaleissa, esimerkiksi ammuksissa. Jos taustana on käytetty mustaa avaruutta,- kuten räiskintäpeleissä usein on tapana, merkkien liikuttelu on varsin vaivatonta.

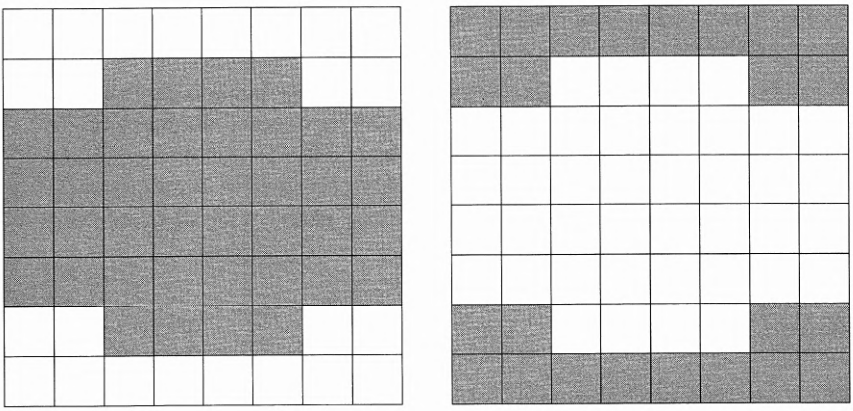
Pistemäisen tai viivamaisen ammuksen liikuttaminen taustagrafiikan päällä vaatii grafiikkamerkkien yhdistelemistä loogisilla operaatioilla. Helpoimmalla

selviää, jos ammuksen grafiikassa on käytetty bittiparia 11. Tällöin liikutus sujuisi seuraavasti:

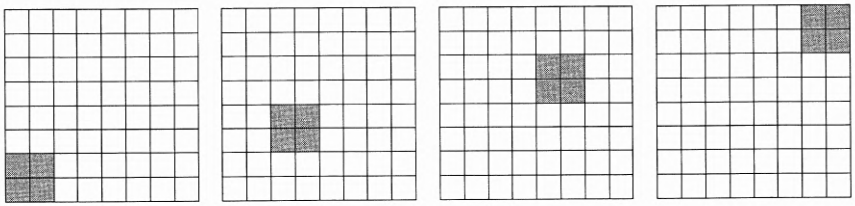
1. Korvataan edellinen ammus taustagrafiikalla.
2. Lasketaan uusi ammuksen paikka.
3. Tutkitaan mikä grafiikkamerkki osuu ammuksen alle
4. Pistetään kuvaruutukoodi muistiin seuraavaa liikutuskertaa varten.
5. Muutetaan ammukseen käytetyn merkin muotoa tekemällä looginen OR-operaatio ammuksen ja taustagrafiikan kanssa.
6. Sijoitetaan ammuksen merkki näytölle.

Käytettäessä monivärisiä liikkuvia hahmoja, operaatio mutkistuu hieman. Tällöin ruudulle sijoitettava merkki on taustagrafiikan, liikkuvan merkin ja maskin yhdistelmä. Maski tarkoittaa grafiikkamerkin "negatiivia", tai paremminkin kuvaa, jossa merkin varjokuvan bitit on käännetty nurin. Jos maskin tekisi monivärisestä merkistä suoraan, sekaan jäisi ylimääräisiä ykkösbittejä.

Aluksi tehdään looginen AND-operaatio taustagrafiikkamerkin ja maskin välillä. Tällöin taustasta jäävät jäljelle vain ne bitit, jotka ovat maskin bittien kohdalla. Tuloksena on merkki, jossa on liikkuvan hahmon muotoinen reikä, aivan kuin taustagrafiikka olisi piparkakkutaikinaa, josta on leikattu palanen pois. Tämän jälkeen merkkiin yhdistetään liikkuvan hahmon bitit loogisella OR-operaatiolla. Näin saadaan aikaan siisti kahden merkin yhdistelmä.



Kuva 6.1 Maskin eli "piparkakkumuotin" luominen.



Kuva 6.2. Viistosti lentävän ammuksen liikeanimaatio grafiikkamerkin sisällä.

Pehmeyttä liikkeisiin

Grafiikkamerkkien siirtely kahdeksaa pikseliä suuremmalla tarkkuudella vaatii pientä kikkailua. Ongelman käytännön ratkaisu riippuu liikkuvan hahmon koosta ja siitä, miten tarkasti sen paikka ja nopeus halutaan määrätä.

Yksinkertaisin tapaus on pistemäinen ammus, jonka leveys ja korkeus on yksi pikseli, monivärigrafiikassa kaksi pikseliä. Ammuksesta tehdään monta versiota, joissa se on hieman eri kohdassa merkin sisällä. Ammusta liikutetaan kahdessa vaiheessa: aluksi näytetään animaatio, jossa liikkeen vaiheet on kuvattu merkin sisällä, sitten siirretään merkkiä ruudulla ja aloitetaan alusta.

Suurempia hahmoja käytettäessä liikkeen välivaiheet eivät enää mahdu yhden merkin sisälle. Liikkeen suunnassa tarvitaan yksi merkki lisää hahmon siirtämistä vasten. Liikeanimaatiossa grafiikkaa tavallaan vieritetään merkistä toiseen. Kun on päästy animaation loppuun, hahmoa siirretään merkin verran ja toistetaan animaatio.

Merkkigrafiikkaa ei kannata käyttää kovin suurissa hahmoissa. Pitää muistaa, että tarvittavien merkkien määrä kasvaa hyvin nopeasti. Esimerkiksi jos hahmoa halutaan liikuttaa pehmeästi kaikkiin suuntiin, jo yhden merkin kokoinen hahmo pitää jakaa neljän merkin kokoiselle alueelle. Merkkien määrä vielä tuplaantuu, jos hahmoja liikutetaan taustan päällä, sillä niistä on oltava muistissa myös maski.

Liikettä bittikartalla

Silloin kun liikkuvat hahmot on toteutettu muilla kuin aidoilla spriteilla, käytetään yleisnimitystä software-spritet eli ohjelmallisesti tuotetut spritet. Merkkigrafiikassa tekniikasta on vain rajallisesti hyötyä. Hieman paremmin se sovel-

tuu bittikarttagrafiikkaan, jossa ei tarvitse pelätä merkkien loppuvan kesken.

Bittikarttaa käytettäessä taustaa ei yleensä vieritetä. Tämän ansiosta ohjelmalle jää niin paljon aikaa, että hahmon liikkeet pystyy laskemaan vaihe vaiheelta. Vaakasuuntainen liike saadaan aikaan pyörittämällä sekä hahmon että sen maskin (eli "piparkakkumuotin") bittejä liikkeen suuntaan. Pystysuuntaista liikettä varten tarvitsee ainoastaan vaihtaa tavua, josta piirtäminen aloitetaan.

Liikkuvan hahmon ja taustan yhdistäminen on hieman vaivalloista, koska bittikartan grafiikka on järjestetty hyvin samantapaisesti kuin merkkien muoto grafiikkamuistissa. Bittikartta rakentuu kahdeksan kertaa kahdeksan pikselin paloista, jotka on ladottu ruudulle vierekkäin. Se on mahdollisimman epäkäytännöllistä grafiikan joustavan käsittelyn kannalta.

Vieritys

Vierityksellä tarkoitetaan sitä, että kuva siirtyy ruudulla pehmeästi, ikään kuin monitori olisi ikkuna, jossa näkyy kerrallaan vain pieni osa suuremmasta alueesta. Kaupallisissa peleissä hyvin toteutettu vieritys on kuin tavaramerkki, joka kertoo ohjelmoijan ammattitaidosta ja tuotteen laadusta.

Vierityksen periaate

Kuusnelosen grafiikkapiirillä on kyky hienosäätää kuvaruudun paikkaa kahdeksan pikselin verran vaaka- ja pystysuunnassa. Ilman tätä ominaisuutta kuvaa voisi siirtää vain merkin verran kerrallaan tai olisi käytettävä erittäin työläitä ja aikaavieviä ohjelmointitekniikoita. Hienosäädön ansiosta liikkeeseen saadaan vaivattomasti myös välivaiheet.

Vieritykseen on käytössä kaksi grafiikkapiirin rekisteriä, $D011 pysty- ja $D016 vaakavierityksselle. Kummassakin muistipaikassa kolme alinta bittiä määrittelevät vierityksen arvon, joka voi olla välillä 0-7. Vaakavierityksessä arvojen kasvattaminen siirtää kuvaa oikealle, pystyvierityksessä alas. Jotta vieritys olisi pehmeää, vieritysrekisterien muuttaminen on tehtävä kerran jokaista kuvaruudun päivitystä kohti. Vieritys onkin näppärintä tehdä rasterikeskeytyksissä.

Kun kuvaa on siirretty vieritysrekisterin avulla seitsemän kertaa, koko kuva on siirrettävä merkin verran vierityksen suuntaan ja rekisteriin palautetaan alkuarvo. Oikealle vieritettäessä seitsemän jälkeen arvoksi pistetään nolla. Vastaavasti vasemmalle vieritettäessä nollan jälkeen tulee seitsemän. Tyypillistä vieritysrutiinia voi kuvailla seuraavasti:

1. Odotetaan rasteririviä heti kuvaruudun alla, normaalisti $FB (251)
2. Muutetaan vieritysrekisteriä.
3. Jos vierityksen arvo ei mahdu välille 0–7, palautetaan alkuarvo ja siirretään kuvaa merkin verran vierityksen suuntaan.

Nopeuden säätäminen

Tähän asti on oletettu, että kuvaa vieritetään pikseli kerrallaan, mutta nopeutta voi myös muutella. Yleensä käytetään nopeuksia välillä 0-8. Kahdeksan pikselin nopeus tarkoittaa, että vierityksen arvo ei muutu, sen sijaan kuvaa siirretään jokaisessa keskeytyksessä. Vieläkin suuremmilla arvoilla kuvaa joutuu välillä siirtämään kahden merkin verran. Nopeudessa voi käyttää myös murtolukuja, esimerkiksi nopeus 0.5 pikseliä saadaan muuttamalla vieritystä vain joka toisessa keskeytyksessä. Tosin tällöin kuvassa on pientä värinää.

Uusi vierityksen arvo saadaan helposti kaavasta:

```
UUSI = (VANHA + NOPEUS) AND 7
```

Kuvaa on siirrettävä, jos VANHA + NOPEUS on suurempi kuin seitsemän tai negatiivinen. AND-käskyllä tuloksesta voi kätevästi napsaista tarvittavat kolme alinta bittiä. Kaavaa voi soveltaa myös silloin, kun nopeudessa käytetään negatiivisia arvoja (kahdeksanbittisissä luvuissa -1 = 255). Negatiivisiin lukuihin perehdytään tarkemmin matematiikkaa käsittelevässä luvussa.

Kuvan kaventaminen

Vaakavierityksessä kuvaa on kavennettava, jotta uusi grafiikka näyttäisi ilmestyvän sulavasti reunuksen alta. Nollaamalla muistipaikan $D016 neljäs bitti kuvan leveydeksi vaihdetaan 38 merkkiä normaalin 40:n sijaan.

Vaakavierityksen normaali arvo on nolla, mikä tarkoittaa käytännössä sitä, että jo kaventamattomassakin kuvassa grafiikka alkaa kadota oikeanpuoleisen reunuksen alle. Kavennetussa kuvassa vasemman reunuksen alla voi olla piilossa seitsemän pikseliä, oikean reunuksen alla saman verran plus yksi kokonainen merkki. Periaatteessa videopiirin ei tarvitsisi kaventaa oikeanpuoleista reunusta lainkaan, mutta se tekee sen kuitenkin. Tämä pieni suunnitteluvirhe on syytä ottaa huomioon vieritystä suunniteltaessa.

Myös pystyvierityksessä kuvaa on pienennettävä. Kuvan normaali korkeus on 25 merkkiä. Nollaamalla muistipaikan $D011 neljäs bitti saadaan korkeudeksi 24 merkkiä. Kuvaa todellakin pienennetään vain yhden merkin verran. Ero vaakavierityksen johtuu siitä, että kuvasta napsaistaan neljä pikseliriviä ylhäältä ja saman verran alhaalta. Tämä on mahdollista siksi, että tavallisessa keskelle ruutua sijoitetussa kuvassa vierityksen alkuarvo on kolme. Kaventamattomassa kuvassa grafiikka katoaa sekä ylä- että alareunuksen alle vieri-

tyksen arvoa muutettaessa.

Vierityksen ongelmat

Vaakavieritys on suhteellisen helppo ohjelmoida. Vieritysrekisterin arvoa on vain vaihdettava sopivassa kohdassa kuvaruutua. Luvun lopussa on esimerkkirutiini, jossa vieritetään kahdeksan rivin korkuista kaistaletta. Aluetta voi helposti laajentaa muuttamalla UPDATE-rutiinia ja keskeytyskohtaa.

Pystyvieritys aiheuttaa enemmän päänvaivaa johtuen kuusnelosen videopiirin rakenteesta. Hankaluuksiin törmätään silloin, kun vain osaa kuvasta käytetään vierityksen ja loput on varattu tulostaululle tai muulle liikkumattomalle grafiikalle.

Kuvan jakaminen pystyvierityksessä on hankalaa siksi, että kuvaa päivittäessään videopiiri ei voi hypätä rasteririvien yli. Pystyvieritystä muutettaessa kuvan puoliskot saattavat kuitenkin mennä päällekkäin, minkä seurauksena alaosa siirtyy merkin verran alaspäin. Vierityksen arvojen ollessa samansuuruisia kuvan alaosa puolestaan on normaalissa kohdassa. Tuloksena on ruman näköisesti ylös ja alas poukkoileva kuva.

Jos kuvan ylempi puolisko on liikkumatonta aluetta, se on vieritettävä niin ylös kuin mahdollista kirjoittamalla rekisterin arvoksi nolla. Vastaavasti jos liikkumaton alue on alareunassa, vierityksen arvoksi kirjoitetaan seitsemän, jotta ruudun alaosa olisi niin alhaalla kuin mahdollista. Nyt ruudulle kuitenkin jää alue, korkeudeltaan 0-7 pikseliä, jossa videopiirin mielestä ei ole lainkaan grafiikkaa. Näille riveille se paremman puutteessa piirtää grafiikkaa, jonka se lukee käytetyn 16:n kilon grafiikkalohkon viimeisestä tavusta. Kuvan puoliskojen vaihtumiskohta on epäesteettinen, joten se olisi syytä piilottaa. Tarkoitukseen soveltuu mainiosti seitsemän vaakasuunnassa laajennettua spritea.

Edellä kuvattu ei ole ainoa tapa jakaa kuva siististi. Kannattaa kokeilla erilaisia vieritysarvoja sekä jakokohdan siirtämistä yhden merkin korkuisella alueella. Tarkalla ajoituksella videopiiriä ehkä onnistuu harhauttamaan ja tuloksena on siisti jakokohta. (Tosin itse en ole siinä vielä onnistunut, mutta eihän se mitään todista.)

Kuvan jakamisen lisäksi on oltava tarkkana siinä, että vieritystä ei muuteta juuri ennen kuvan yläreunaa. Väliä on jätettävä vähintään kahdeksan rasteririviä, mieluummin vieläkin enemmän. Pystyvierityksen muuttaminen liian lähellä kuvan alkua sekoittaa grafiikan päivityksen vaakasuunnassa. Tästä pienestä videopiirin heikkoudesta on yllättäen myös hyötyä, kuten käy ilmi erikoistehosteista kertovassa luvussa.

Vapaasuuntainen vieritys

Vaaka- ja pystyvieritystä yhdistämällä saa aikaan monenlaisia vieritystyylejä. Loivasti yläviistoon vierivää kuvaa käytti esimerkiksi vanha räiskintäklassikko Zaxxon. Kahdeksansuuntaisessa vierityksessä kuvaa puolestaan liikutetaan joystickin liikkeitä myötäillen vaaka-, pysty- ja viistosuunnissa. Jos taas vaaka- ja pystyvierityksissä käytetään vaihtuvia nopeuksia, puhutaan vapaasuuntaisesta vierityksestä. Sen avulla saadaan aikaan sulavasti ilman välivaiheita suuntaa muuttava vieritys.

Vapaasuuntaisessa ja muissa yhdistetyissä vierityksissä on muistettava sekä vaaka- että pystyvieritykseen liittyvät seikat. Kuva on kavennettava kummassakin suunnassa ja kuvan jakaminen kahteen osaan on tehtävä samoin kuin pystyvierityksessä.

$D011	Bitit 0-2: pystyvierityksen arvo
	Bitti 3: 0 = 24 riviä, 1 = 25 riviä
$D016	Bitit 0-2: vaakavierityksen arvo
	Bitti 3: 0 = 38 merkkiä, 1 = 40 merkkiä

Taulukko 8.1. Vierityksessä tarvittavat rekisterit ja bittien käyttötarkoitus.

Grafiikkaa muistista ruudulle

Vierityksessä kuvan sisältö muuttuu jatkuvasti, joten ohjelman on luettava muistista uutta grafiikkaa. Sen siirtämisessä ruudulle on useita tapoja: reunuksien päivitys, grafiikkapuskuri ja kaksoispuskurointi. Kaikille löytyy käyttöä ja ohjelmoijan on päätettävä mikä kulloinkin tuntuu parhaalta.

Reunuksien päivityksessä koko kuva siirretään vierityksen suuntaan ja uutta grafiikkaa siirretään reunuksien alle. Vapaasuuntaista vieritystä käytettäessä on otettava huomioon, että kuvaa joutuu siirtämään kahdeksaan eri suuntaan, siis myös viistoon. Menetelmän etuna on se, ettei ole niin väliksi missä muodossa grafiikka on tallennettu muistiin, kunhan päivitysrutiini osaa kaivaa sen esiin. Taidokkaalla grafiikan pakkaamisella ruudulle saa erittäin laajoja vieriviä alueita. Haittana on taustagrafiikan käsittelyn jähmeys. Grafiikkaa on hyvin työläs muutella ohjelman aikana, varsinkin jos pelialueella voi liikkua vapaasti edes takaisin. Esimerkiksi merkkigrafiikalla toteutettujen rakennusten räjähtäminen raunioiksi ja muu vastaava on tehtävä sekä suoraan kuvaruudulle että grafiikka-

taulukoihin.

Jos taustagrafiikkaa on muuteltava ahkerasti, on kätevintä käyttää grafiikkapuskuria. Ero edelliseen on siinä, että vieritettävä alue on tallennettu muistiin pakkaamattomassa muodossa ja koko kuvaruutu piirretään jokaisella päivityskerralla. Erilliset vieritettävät alueet voivat olla tiiviiksi pakattuja, mutta ennen kullekin pelialueelle siirtymistä ohjelma purkaa grafiikan puskurialueelle, josta se sitten luetaan kuvaruudulle. Animaatiota, liikkuvia merkkejä ja muita vastaavia tarvitsee käsitellä ainoastaan puskurialueella, ei suoraan kuvaruudulla.

Puskurointi on erityisen kätevää, jos alueen jokaisella vaakarivillä on leveyttä 256 merkkiä (n. 6 1/2 kuvaruutua). Nyt muistiosoitteesta näkee suoraan missä kohtaa aluetta ollaan: alatavu kertoo pystysarakkeen ja ylätavu vaakarivin. Grafiikkapuskurin käyttö helpottaa suuresti vapaasuuntaisen vierityksen ohjelmointia, sillä vieritysrutiinin ei tarvitse siirrellä kuvaa moneen eri suuntaan, tarvitsee vain tietää mistä kohtaa muistia grafiikan siirtäminen kuvaruudulle aloitetaan.

Kuvan päivitystavoista monimutkaisin ja rajoittavin mutta samalla tehokkain on kaksoispuskurointi. Sitä voi käyttää vain silloin, kun kuvaa vieritetään samaan suuntaan vakionopeudella, joka on korkeintaan neljä pikseliä keskeytystä kohden. Ideana on käyttää kahta kuvaruutua, joita näytetään vuorotellen. Kuvan liikkuessa vieritysrekisterien varassa ohjelma piirtää valmiiksi pala kerrallaan seuraavaa kuvaa, jossa grafiikka on siirtynyt merkin verran. Esimerkiksi yhden pikselin vieritysnopeudella kuvan piirtämisen voi jakaa kahdeksaan päivityskertaan. Viimeisellä kerralla piilossa ollut kuva otetaan näkyviin. Käytännössä se tapahtuu muuttamalla muistipaikan $D018 arvo vastaamaan uuden kuvaruudun sijaintia muistissa. Sitten aloitetaan kierros alusta ja aletaan siirtää grafiikkaa ensimmäiseen kuvaruutuun, joka on nyt piilossa.

Yhteenveto erilaisista tavoista siirtää grafiikkaa kuvaruudulle

Reunuksien päivitys:

* Siirretään koko kuva merkin verran vierityksen suuntaan.

* Siirretään reunuksen alle uutta grafiikkaa.

Grafiikkapuskuri:

* Jos grafiikka on pakattu, se puretaan grafiikkapuskuriin ennen vierityksen aloittamista.

* Luetaan kuvan koko sisältö grafiikkapuskurista jokaisella päityskerralla.

Kaksoispuskuri:

* Varataan muistista tilaa kahdelle kuvaruudulle.

* Jokaisella päivityskerralla piirretään valmiiksi osa piilossa olevasta kuvaruudusta.

* Kun kuvaa on vieritetty merkin verran, otetaan näkyviin piilossa ollut kuva ja aletaan päivittää näkymättömäksi jäävää kuvaa.

Esimerkki vaakavierityksestä

Seuraavassa esimerkissä vieritetään kuvan kahdeksaa ylintä merkkiriviä. Vieritettävää aluetta laajennettaessa on muutettava keskeytyskohtaa, jotta ruudun jako tehtäisiin oikeassa kohdassa. Alunperin kohdaksi on määritelty yhdeksännen merkkirivin alku, eli rasteririvi $72 (114). Oikean kohdan voi laskea kaavalla:

```
KESKEYTYSKOHTA = MERKKIRIVIEN MÄÄRÄ X 8 + 50
```

Rutiinissa on käytetty kahta keskeytysrutiinia. Ensimmäisessä kuva palautetaan normaaliksi, lasketaan uusi vierityksen arvo ja päivitetään vierivän alueen grafiikka. Kuvan alle ajoitetussa keskeytyksessä muutetaan vieritysrekisteriä. Mikäli alue on tarpeeksi korkea, ei tarvita kahta keskeytysrutiinia, sillä ruudun päivityksen jälkeen ollaan jo saavuttu kuvan alareunaan. Näin ollen toisen rutiinin voi liittää suoraan ensimmäisen perään.

Kuvan sisältö luetaan grafiikkapuskurista, joka on sijoitettu muistialueelle $4000–$47FF. Kun vieritettävän alueen korkeutta lisätään, jokainen uusi rivi kasvattaa puskurin kokoa 256:lla tavulla.

```
        ORG $1000
START   SEI          ;alustetaan keskeytys
        LDA #$7F     ;IRQ pois päältä
        STA $DC0D
        LDA #$35     ;kernal pois päältä
        STA 1
        LDA #<RAS1   ;keskeytysvektori
        STA $FFFE
        LDA #>RAS1
        STA $FFFF
        LDA #1       ;rasterikeskeytys päälle
        STA $D01A
        LDA #$72     ;keskeytyskohta = $72
        STA $D012
        LDA #$1B
        STA $D011
        LDA $DC0D    ;kuitataan IRQ
        CLI
MAIN    JMP MAIN     ;pyöritään paikallaan...
*  Keskeytys kohdassa $72:
*  Vieritysnopeus ja kuvan päivitys.
RAS1    STA REMA+1   ;rekisterit talteen
        STX REMX+1
        STY REMY+1
        LDA $D019    ;kuitataan keskeytys
        STA $D019
        LDX #4       ;ajoitetaan vaihtumiskohta
ILOOP1  DEX          ;reunuksen alle piiloon
        BNE ILOOP1
        LDA #0       ;nollataan X-vieritys
        STA $D016
        JSR JOY      ;peliohjain ja vierityksen arvo
        JSR UPDATE   ;ruudun päivitys
        LDA #$FB     ;seuraava keskeytys kohdassa $FB
        STA $D012
        LDA #<RAS2   ;uusi hyppyvektori
        STA $FFFE
```

```
        LDA #>RAS2
        STA $FFFF
REMA    LDA #0
REMX    LDX #0
REMY    LDY #0
        RTI
*  Keskeytys kohdassa $FB:
*  Päivitetään vieritysrekisteri.
RAS2    STA REMA+1   ;rekisterit talteen
        STX REMX+1
        STY REMY+1
        LDA $D019    ;kuitataan keskeytys
        STA $D019
        LDA SCROLL   ;vierityksen arvo rekisteriin
                     $D016
        STA $D016
        LDA #$72     ;seuraava keskeytys kohdassa $72
        STA $D012
        LDA #<RAS1   ;uusi hyppyvektori
        STA $FFFE
        LDA #>RAS1
        STA $FFFF
        JMP REMA     ;pois keskeytyksestä
*  Peliohjain ja vierityksen arvo
JOY     INC IRQCNT
        LDA IRQCNT
        AND #3
        BNE JLOOP3   ;luetaan ohjain vain joka
                     ;4. kerralla
        LDX XSPEED
        LDA $DC00
        TAY
        AND #8       ;ohjain oikealle?
        BEQ JLOOP1
        TYA
        AND #4       ;ohjain vasemmalle?
        BNE JLOOP3
        INX          ;lisätään vieritysnopeutta
```

```
        CPX #8        ;suurin sallittu arvo = 8
        BEQ JLOOP3
        BNE JLOOP2
JLOOP1  DEX           ;vähennetään vieritysnopeutta
        CPX #$F8      ;pienin sallittu arvo = $F8
        BEQ JLOOP3
JLOOP2  STX XSPEED
JLOOP3  CLC
        LDA SCROLL    ;vieritysrekisteri
        AND #7
        ADC XSPEED    ;lisätään nopeus
        TAX
        AND #7
        STA SCROLL    ;uusi vierityksen arvo
        TXA
        AND #$F8      ;tapahtuiko ylivuoto?
        BMI JLOOP4
        BNE JLOOP5
        RTS
JLOOP4  LDA PLAYF     ;siirretään ruutua vasemmalle
        CMP #255-38   ;ollaanko reunassa?
        BEQ STOP2
        INC PLAYF
        RTS
JLOOP5  LDA PLAYF     ;siirretään ruutua oikealle
        BEQ STOP      ;ollaanko reunassa
        DEC PLAYF
        RTS
STOP    LDA #$17      ;pysäytetään vieritys
        .BYTE $2C     ;$2C + LDA #$10 = BIT $10A9
STOP2   LDA #$10
        STA SCROLL
        LDA #0
        STA XSPEED
        RTS
SCROLL  .BYTE 0       ;X-vierityksen arvo
XSPEED  .BYTE $FF     ;nopeus
```

```
PLAYF  .BYTE 0      ;grafiikkapuskurin osoitin
IRQCNT .BYTE 0      ;laskuri
*  Kuvaruudun päivitys.
UPDATE LDA PLAYF    ;grafiikkaosoitin
       STA LINE1+1 ;siirretään osoitteen alatavuksi
       STA LINE2+1
       STA LINE3+1
       STA LINE4+1
       STA LINE5+1
       STA LINE6+1
       STA LINE7+1
       STA LINE8+1
       LDX #$27
LINE1  LDA $4000,X ;päivitetään 8 riviä
       STA $0400,X
LINE2  LDA $4100,X
       STA $0428,X
LINE3  LDA $4200,X
       STA $0450,X
LINE4  LDA $4300,X
       STA $0478,X
LINE5  LDA $4400,X
       STA $04A0,X
LINE6  LDA $4500,X
       STA $04C8,X
LINE7  LDA $4600,X
       STA $04F0,X
LINE8  LDA $4700,X
       STA $0518,X
       DEX
       BPL LINE1
       RTS
       END
```

Musiikki ja äänitehosteet

Hyvästäkin pelistä jää ankea vaikutelma jos siinä ei ole lainkaan ääniä. Varsinkin toimintapeleissä äänitehosteet ovat hyvin tärkeä osa pelin tunnelmaa. Korkeintaan ajattelua vaativissa strategiapeleissä äänipuoleen ei tarvitse kiinnittää niin paljon huomiota, tai ne voi jättää kokonaan pois.

Kuusnelosen äänipiiri (SID eli Sound Interface Device) on sukupolvensa parhaita. Käytössä on kolmc äänikanavaa ja neljä aaltomuotoa, joiden avulla taitava ohjelmoija saa aikaan melkein mitä ääniä tahansa. Äänipiirin ohjelmointi ei kuitenkaan ole niitä kaikkein helpoiten hallittavia asioita. Toisin kuin esimerkiksi grafiikassa, äänipiirin sisältöä on jatkuvasti muuteltava, jotta saataisiin haluttu lopputulos.

Äänipiirin rakenne

Muistipaikasta $D400 alkavat äänirekisterit ovat write only -rekistereitä, toisin sanoen niihin voi kirjoittaa, mutta niiden sisältöä ei voi lukea. Tähän on pari poikkeusta, joihin palataan myöhemmin. Mikäli ohjelman on tiedettävä mitä äänirekistereihin on kirjoitettu, on pakko tallentaa arvot "haamurekistereihin". Niillä tarkoitetaan yksinkertaisesti muuttujina käytettäviä muistipaikkoja, joiden sisältö siirretään äänirekisteriin oikealla hetkellä.

Kaikkien kolmen äänikanavan rekisterit ovat rakenteeltaan identtisiä. Ne alkavat muistipaikoista $D400, $D407 ja $D40E, yhdellä kanavalla on siis käytössä seitsemän rekisteriä. Lisäksi on käytössä neljä rekisteriä ($D415–$D418), jotka vaikuttavat kaikkiin äänikanaviin. Näistä tärkein on muistipaikka $D418, jonka neljä alinta bittiä määräävät kaikkien äänien voimakkuuden.

Aaltomuoto

Äänet syntyvät ilmassa etenevistä paineaalloista. Ne voidaan kuvata graafisesti, jolloin saadaan näkyviin äänen aaltomuoto. Täysin "puhdas" ääni kuvataan tasaisesti aaltoilevana sinikäyränä, joka muistuttaa kyljelleen käännettyä S-kirjainta. Ääniaallon peräkkäisten huippujen välinen etäisyys määrää äänen taajuuden. Mitä tiheämmin aalto väreilee, sitä korkeammalta ääni kuulostaa. Ääniaallon muoto puolestaan vaikuttaa ratkaisevasti äänen sointiin. Puhdas sinikäyrä on "perusääni", jonka kuulet esimerkiksi nostaessasi puhelimen

luurin korvallesi.

Kuusnelosessa on neljä valmiiksi määriteltyä aaltomuotoa. Lähimpänä puhdasta sinikäyrää on kolmioaalto:

Siniaallosta poiketen sen sointi on jonkin verran ontto ja huilumainen. Sama ilmaistuna tieteellisemmin: äänellä on pariton yläsävelsarja. Kolmioaallolla tuotetut äänet ovat hyvin hentoja, minkä vuoksi ne hukkuvat helposti muiden äänien alle.

Saha-aalto on hieman kolmioaallon näköinen:

Saha-aallon sointi on kolmioaaltoa huomattavasti rosoisempi, johtuen täyteläisestä yläsävelsarjasta. Äänen sointi tuo lähinnä mieleen viulun, trumpetin ja muut säröiseltä kuulostavat soittimet.

Kolmas aaltomuoto on pulssi- eli neliöaalto:

Pulssiaalto on siitä mielenkiintoinen, että äänen laatuun voi vaikuttaa muuttamalla pulssin leveyttä. Aallon leveyden määrittelyyn tarvitaan kullekin kanavalle kaksi rekisteriä, joista on käytössä 12 bittiä. Jos rekistereiden arvona on $0800, pulssin leveys on tasan puolet aallon kokonaispituudesta. Tällöin ääni kuulostaa hyvin lavealta, hieman klarinetin tapaiselta. Pulssia leventämällä tai kaventamalla ääni muuttuu kireämmäksi, säkkipillin tapaan soivaksi. Kun pulssia vaihtaa nopeasti leveästä kapeaan ja taas takaisin, saadaan aikaan hyvin kaunis syntetisaattorimainen ääni.

Edellä mainittujen puhtaiden aaltomuotojen lisäksi äänipiiri pystyy tuottamaan kohinaa. Kohinalla ei varsinaisesti ole kiinteää aaltomuotoa, sillä ääni tuotetaan satunnaisgeneraattorilla. Näin ollen kohinalla ei myöskään ole kiinteää taajuutta. Äänen korkeus on keskiarvo hyvin laajalla taajuusalueella poukkoilevista äänistä. Kohinalla saa aikaan tuulen suhinaa, askelten kopinaa, räjähdyksiä ja muita epämääräisiä ääniä.

Äänikanavien aaltomuodot valitaan seuraavilla rekistereillä:

$D404 Kanavan 1 aaltomuoto

$D40B Kanavan 2 aaltomuoto

$D412 Kanavan 3 aaltomuoto

Aaltomuodon valinnassa käytetään rekisterien neljää ylintä bittiä:

Bitti 4: kolmioaalto

Bitti 5: saha-aalto

Bitti 6: pulssiaalto

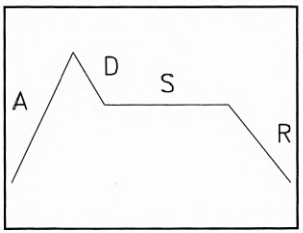
Bitti 7: kohina

Yksi äänikanava pystyy käyttämään myös useita aaltomuotoja päällekkäin. Esimerkiksi kytkemällä päälle sekä kolmio- että saha-aallon tuloksena on hyvin erikoinen sirkkelimäinen, ohut ääni. Kohina on poikkeus. Jos sitä käyttää yhdessä jonkun muun aaltomuodon kanssa, äänikanava saattaa lukkiutua.

Verhokäyrä eli ADSR

Verhokäyrällä tarkoitetaan äänen voimakkuuden vaihtelua. Esimerkiksi rummun ääni nousee nopesti huippuarvoon ja sitten putoaa nollaan, sitä hitaammin mitä enemmän rummussa on sointia ja huoneessa kaikua. Viulussa taas ääni kohoaa aluksi huippuunsa, sitten voimakkuus laskee hieman ja pysyy vakiotasolla viulistin liikuttaessa joustaan. Jousen irrotessa viulunkielestä ääni putoaa nollaan hyvin nopeasti. Äärimmäinen esimerkki on sähköurut, jossa ääni nousee heti huippuun ja pysyy siellä niin kauan kuin sormea pidetään koskettimella. Lopuksi ääni putoaa välittömästi nollaan.

Kuusnelosen äänipiirissä verhokäyrä on jaettu neljään osaa: attack, decay, sustain ja release. Attack tarkoittaa sitä, miten nopeasti ääni kohoaa nollasta huippuarvoon. Decayn aikana taas ääni putoaa huipusta vakio- eli sustain-tasolle. Release tarkoittaa äänen putoamista vakiotasolta nollaan.



Kaavio 9.1 Verhokäyrän osat: A = attack, D = decay, S = sustain, R = release.

Kaikille verhokäyrän osille määritelään arvot väliltä 0–15. Verhokäyrän attack-, decay- ja release-jaksoille määrätyt arvot tarkoittavat pituutta, sustain taas tulkitaan äänen voimakkuudeksi. Tietoa tarvitaan vain neljä bittiä, joten yhtä kanavaa kohti on käytössä kaksi äänirekisteriä (taulukko 9.3).

Attack-jakso alkaa kun äänikanavan gate-bitti kirjoitetaan ykköseksi. Decay-jakson jälkeen äänenvoimakkuus pysyy sustain-tasolla, kunnes gate-bitti nollataan. Sen jälkeen alkaa release-jakso. Sustain-jakson pituutta varten ei ole omaa rekisteriä. Ohjelman on yksinkertaisesti odoteltava jonkin aikaa ennen kuin se nollaa gate-bitin.

Periaatteessa gate-bitin voi nollata jo attack-jakson aikana, esimerkiksi jos haluaa hyvin terävästi paukahtavia ääniä. Samoin uuden attack-jakson voi aloittaa ennen kuin edellinen release-jakso on ehtinyt loppuun. Peräkkäisten äänien verhokäyrissä voi siis olla päällekkäisyyttä ja niitä voi kontrolloida ohjelmallisesti melko vapaasti. Hyvin nopeasti toisiaan seuraavissa äänissä saattaa kuitenkin esiintyä häiriötä, esimerkiksi uuden attack-jakson käynnistys voi epäonnistua. Tällöin kannattaa kokeilla erilaisia verhokäyriä, esimerkiksi pienentää sustainin arvoa tai nollata se kokonaan.

Arvo	Attack-aika	Decay/release-aika
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	1.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

Taulukko 9.1 Verhokäyrän osien kesto eri arvoilla (ms = millisekunti eli 1/1000 sekuntia).

Äänen korkeus

Kaikilla äänikanavilla on käytössä kaksi rekisteriä äänen korkeutta varten, toisin sanoen taajuuden asetukseen käytetään 16-bittisiä lukuja. Rekistereiden arvoa ei tulkita suoraan äänen taajuudeksi. Tämän voi päätellä jo siitäkin, että suurin mahdollinen arvo olisi yli 65000, kun taas ihmiskorva kuulee parhaimmillaan taajuuden 15000–20000 Hz.

Äänipiirin taajuusalue on noin 0–4000 Hz. Se vastaa kahdeksaa oktaavia, eli sävelaluetta C-0 – H-7. Yksittäisen sävelen taajuuden voi määritellä erittäin tarkasti, mikä mahdollistaa sävelen huojuttamisen (vibrato), liukumiset sävelestä toiseen (portamento) ja kauniin chorus-efektin, jossa samaa säveltä soitetaan usealla kanavalla aavistuksen verran eri korkeudelta.

Musiikin tekemistä varten on taulukoitava muistiin kaikki nuotit. Taulukossa 9.4 on luettelo nuotteja vastaavista taajuusasetuksista. On hyvä tietää, jos ei muuten niin yleissivistyksen kannalta, että taulukossa on käytetty tasavireistä asteikkoa. Kuitenkin täysin puhtaassa musiikissa asteikko on viritettävä erikseen kaikille sävellajeille. Käytännössä tällä on merkitystä vain jos musiikintekijällä on erittäin hyvä sävelkorva ja hän tavoittelee täydellisyyttä. Pianossa on yleensä tasavireinen asteikko, kun taas esimerkiksi joillain puhaltimilla saa aikaan täysin puhtaita säveliä.

Äänen suodatus

Suodattimella voi vaimentaa tietylle taajuusalueelle osuvia ääniä. Varsinkin kohinassa suodatuksella on varsin dramaattinen vaikutus äänen laatuun. Rekisterin $D417 biteillä 0–2 voi valita, mihin äänikanaviin suodatin vaikuttaa.

Suodattimen vaikutus äänen riippuu kahdesta seikasta, katkaisutaajuudesta ja suodattimen tyypistä. Katkaisutaajuus kirjoitetaan muistipaikkoihin $D415 ja $D416. Äänentaajuudesta poiketen se on 11-bittinen luku, sillä alemmassa tavussa vain biteillä 0-2 on merkitystä. Katkaisutaajuudesta riippuu, minkä taajuusalueen ääniä vaimennetaan. Käytössä on kolme erilaista suodatinta, jotka valitaan muistipaikan $D418 biteillä 4–6:

– low-pass-suodatin (bitti 4) vaimentaa katkaisutaajuutta korkeammat äänet

– band-pass-suodatin (bitti 5) vaimentaa katkaisutaajuutta korkeammat ja matalammat äänet

– high-pass-suodatin (bitti 6) vaimentaa katkaisutaajuutta matalammat äänet

Suodattimia voi myös yhdistellä, esimerkiksi low- ja high-pass-suodattimien yhdistelmällä suodatetaan vain katkaisutaajuutta lähellä olevat äänet.

Suodatettuihin ääniin voi lisätä resonanssia muistipaikan $D417 biteillä 4–7. Näiden neljän bitin avulla resonanssin voimakkuuden voi valita väliltä 0–15. Resonanssi vahvistaa katkaisutaajuuden lähellä olevia ääniä, jolloin ääni kuulostaa hieman säröiseltä.

Suodattimen käyttöä hankaloittaa joidenkin vanhempien kuusnelosten äänipiireissä oleva valmistusvirhe. Käytännössä se ilmenee siten, että katkaisutaajuudella on hyvin sattumanvarainen vaikutus. Suodatettu ääni kuulostaa täysin erilaiselta eri koneissa.

Rengasmodulaatio ja synkronointi

SID pystyy tuottamaan kahden äänen yhdistelmiä, joilla saadaan aikaan varsin eksoottisia tehosteita. Rengasmodulaatiossa ja synkronoinnissa tarvitaan aina kaksi kanavaa, joista toinen yleensä kytketään hiljaiseksi ja toisesta kuuluu äänipiirin muokkaama lopputulos. Esimerkiksi kun käytetään kanavaa 1, kanava 3 on tämä hiljainen osapuoli.

Rengasmodulaatiolla saadaan aikaan mutkikas kahden ääniaallon yhdistelmä, jonka taajuus on epämääräinen ja sointi epäharmoninen. Tuloksena voi olla vaikkapa kirkonkelloja muistuttava äänitehoste. Rengasmodulaatio kytketään päälle kontrollirekisterin bitillä 2. Moduloitavassa äänessä on käytettävä kolmioaaltoa, jotta modulaatiolla olisi jotain vaikutusta. Äänen sointiin vaikuttaa muokkaukseen käytettävän äänen taajuus.

Toinen tapa yhdistää kaksi ääniaaltoa on synkronointi. Se tarkoittaa loogista AND-operaatiota kahden ääniaallon välillä. Lopputuloksena voi olla hyvinkin erikoisia harmonisia rakenteita. Synkronointi valitaan kontrollirekisterin bitillä 1. Muokkaukseen käytettävän äänikanavan asetuksista ainoastaan taajuudella on vaikutusta äänen laatuun. Lisäksi muokattavan äänen taajuus on mielellään oltava korkeampi kuin muokkaavalla äänellä.

Luettavat rekisterit

Äänipiirissä on kaksi rekisteriä, joita voi ainoastaan lukea (read only -rekisteri): $D41B ja $D41C. Kummatkin liittyvät kanavan 3 ulostuloon. Rekisteristä $D41B voi lukea kanavan aaltomuodon ja muistipaikasta $D41C verhokäyrän.

Soittorutiinissa näitä lukuja voi käyttää äänien muokkaamiseen. Esimerkiksi jos käytössä on kolmioaalto, rekisterin $D41B arvo alkaa nollasta, nousee tasaisesti 255:een ja putoaa jälleen tasaisesti nollaan. Jakson nopeus riippuu suoraan äänen korkeudesta. Peliohjelmoijalle rekisteristä on eniten hyötyä silloin, kun

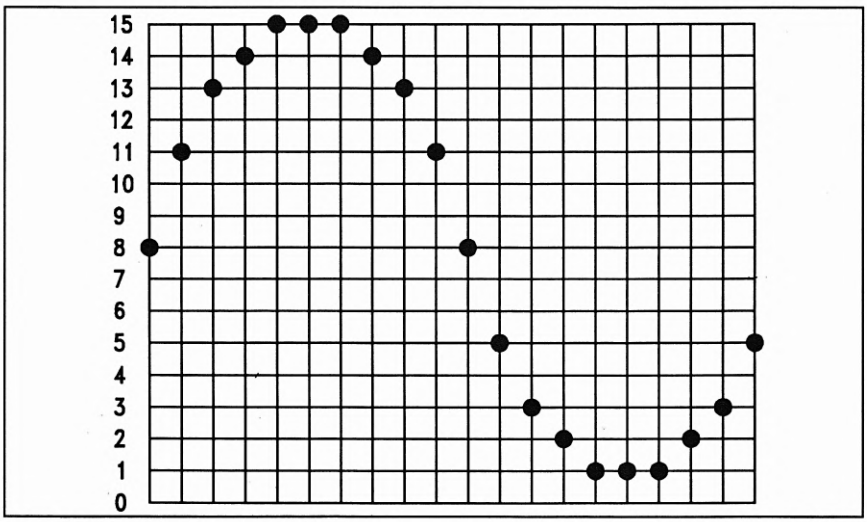
kolmannen kanavan aaltomuodoksi on valittu kohina. Tällöin rekisteristä voi lukea satunnaislukuja väliltä 0-255.

Digitoidut äänet

Kuusnelosessa on mahdollista soittaa myös digitoituja ääniä. Tämä perustuu siihen, että aina kun äänenvoimakkuutta muuttaa, kuuluu pieni napsahdus. Mitä suurempi muutos, sitä kovempi ääni. Uudemmissa kuusnelosissa on paranneltu äänipiiri, jossa tämä napsahdus on niin vaimea, että digitoidun äänen tuottaminen on vaikeaa.

Peräkkäisillä napsahduksilla on mahdollista luoda hyvin monimutkaisia ääniaaltoja. Tekniikasta käytetään nimitystä samplaaminen. Rekisterissä $D418 äänenvoimakkuudelle on varattu neljä alinta bittiä, joten kuusnelosella voi soittaa nelibittisiä sampleja. Peräkkäisen napsahduksen nopeutta kutsutaan puolestaan samplaus- eli näytteenottotaajuudeksi. Ääni kuulostaa sitä puhtaammalta mitä suurempaa taajuutta käytetään. Käytännössä riittää 8 kilohertsin taajuus, toisin sanoen voimakkuutta muutetaan 8000 kertaa sekunnissa.

Taulukossa 9.2 on havainnollistettu ääniaallon muodostumista. Siinä on luotu tasaisesti värähtelevä siniaalto kahdellakymmenellä äänenvoimakkuuden muutoksella. Asian voisi ilmaista niinkin, että aalto muodostuu kahdestakym-



Taulukko 9.2 Esimerkki samplatusta sinikäyrästä. Ääniaallon perustasoksi eli nollakohdaksi voidaan valita voimakkuus 7 tai 8.

menestä samplesta eli näytteestä.

Seuraava rutiini soittaa taulukossa kuvattua ääniaaltoa:

```
PLAY   SEI          ;estetään keskeytykset
       LDA #$0B
       STA $D011    ;sammutetaan kuvaruutu
LOOP1  LDX #$00
LOOP2  LDA SINI,X   ;luetaan näyte taulukosta
       STA $D418    ;kirjoitetaan äänenvoimakkuudeksi
       LDY #$0C     ;viivesilmukka
DELAY  DEY
       BNE DELAY
       INX
       CPX #20      ;kaikkiaan 20 näytettä
       BEQ LOOP1    ;äänen alkuun
       BNE LOOP2    ;seuraava näyte
SINI   .BYTE 8,11,13,14,15,15,15,14,13,11
       .BYTE 8,5,3,2,1,1,1,2,3,5
PULSE  .BYTE 0,0,0,0,0,0,0,0,0,0
       .BYTE 15,15,15,15,15,15,15,15,15,15
```

Kirjoita rutiini ja kokeile mitä tapahtuu. Kaiuttimesta pitäisi kuuluu puhdas perusääni. Seuraavaksi muuta rutiinia sen verran, että näyte luetaan taulukosta PULSE. Tuloksena on tyypillinen pulssiaallon synnyttämä ääni. Äänen korkeutta voi säätää muuttamalla viiveen pituutta. Normaalisti sampleja ei tietenkään kirjoiteta koneeseen käsin, vaan ne luodaan äänendigitoijalla.

Esimerkissä sammutettiin kuvaruutu, jotta samplea soitettaisiin mahdollisimman tasaisesti. Jos kuva on päällä, videopiiri ryöstää prosessoriaikaa aiheuttaen lievää heittelyä soittorutiinin kestossa. Ääni kuulostaa säröisemmältä ja myös sen taajuus voi hieman muuttua.

Samplattuja rumpuja sekä muita epäharmonisia soittimia on helpointa käyttää, sillä niiden äänenkorkeutta ei tarvitse säätää kovin tarkasti. Niitä voi soittaa hyvin yksinkertaisella silmukalla. Jos taas halutaan luoda digitoitua musiikkia, samplet on viritettävä oikealle korkeudelle. Se onnistuu parhaiten käyttämällä NMI-keskeytyksiä. Erikorkuiset äänet luodaan keskeytystaajuutta muuttamalla. Ohjelmointitekniikan etuna on koneelle jäävä ylimääräinen aika. On jopa mahdollista käyttää digitoituja ääniä pelin taustalla, mutta tällöin on karsittava pois vieritys ynnä muut aikaa vievät tehosteet.

Taulukko 9.3 Äänipiirin rekisterit.

Kanava 1:

$D400 Äänen korkeus, alatavu
$D401 Äänen korkeus, ylätavu
$D402 Pulssiaallon leveys, alatavu
$D403 Pulssiaallon leveys, bitit 8-11 (ylätavun puolikas)
$D404 Kontrollirekisteri:
Bitti 0: gate-bitti
1 = aloita attack-jakso
0 = aloita release-jakso
Bitti 1: 1 = synkronoi kanavan 3 kanssa
Bitti 2: 1 = rengasmodulaatio kanavan 3 kanssa
Bitti 3: 1 = sammuta kanava ("test bit")
Bitti 4: 1 = kolmioaalto
Bitti 5: 1 = saha-aalto
Bitti 6: 1 = pulssiaalto
Bitti 7: 1 = kohina
$D405 Verhokäyrän attack ja decay:
Bitit 0-3: decay
Bitit 4-7: attack
$D406 Verhokäyrän sustain ja release:
Bitit 0-3: release
Bitit 4-7: sustain

Kanava 2:

$D407 Äänen korkeus, alatavu
$D408 Äänen korkeus, ylätavu
$D409 Pulssiaallon leveys, alatavu
$D40A Pulssiaallon leveys, bitit 8-11 (ylätavun puolikas)
$D40B Kontrollirekisteri:
Bitti 0: gate bitti
1 = aloita attack-jakso
0 = aloita release-jakso

Bitti 1: 1 = synkronoi kanavan 1 kanssa
Bitti 2: 1 = rengasmodulaatio kanavan 1 kanssa
Bitti 3: 1 = sammuta kanava ("test bit")
Bitti 4: 1 = kolmioaalto
Bitti 5: 1 = saha-aalto
Bitti 6: 1 = pulssiaalto
Bitti 7: 1 = kohina

$D40C Verhokäyrän attack ja decay:
Bitit 0-3: decay
Bitit 4-7: attack

$D40D Verhokäyrän sustain ja release:
Bitit 0-3: release
Bitit 4-7: sustain

Kanava 3:

$D40E Äänen korkeus, alatavu
$D40F Äänen korkeus, ylätavu
$D410 Pulssiaallon leveys, alatavu
$D411 Pulssiaallon leveys, ylätavun puolikas (bitit 8-11)
$D412 Kontrollirekisteri:
Bitti 0: gate-bitti
1 = aloita attack-jakso
0 = aloita release-jakso
Bitti 1: 1 = synkronoi kanavan 2 kanssa
Bitti 2: 1 = rengasmodulaatio kanavan 2 kanssa
Bitti 3: 1 = sammuta kanava ("test bit")
Bitti 4: 1 = kolmioaalto
Bitti 5: 1 = saha-aalto
Bitti 6: 1 = pulssiaalto
Bitti 7: 1 = kohina

$D413 Verhokäyrän attack ja decay:
Bitit 0-3: decay
Bitit 4-7: attack

$D414 Verhokäyrän sustain ja release:

Bitit 0-3: release

Bitit 4-7: sustain

Kaikki kanavat:

$D415 Suodattimen taajuus, alatavun puolikas (bitit 0-2)

$D416 Suodattimen taajuus, ylätavu

$D417 Suodattimen kontrollirekisteri:

Bitti 0: 1 = suodata äänikanava 1

Bitti 1: 1 = suodata äänikanava 2

Bitti 2: 1 = suodata äänikanava 3

Bitti 3: 1 = suodata ulkoisesti syötetty ääni

Bitit 4-7: suodatuksen resonanssi (0-15)

$D418 Äänen voimakkuus ja suodattimen tyyppi:

Bitit 0-3: äänen voimakkuus (0-15)

Bitti 4: 1 = vaimenna suodatustaajuutta korkeammat äänet (low-pass)

Bitti 5: 1 = vaimenna suodatustaajuutta korkeammat ja matalammat äänet (band-pass)

Bitti 6: 1 = vaimenna suodatustaajuutta matalammat äänet (high-pass)

Bitti 7: 1 = katkaise kanavan 3 ulostulo

Taulukko 9.4 Nuotit ja niitä vastaavat taajuusasetukset.

Nuotti	Desimaalina	Heksana
C-0	268	$010C
C#-0	284	$011C
D-0	301	$012D
D#-0	318	$013E
E-0	337	$0151
F-0	358	$0166
F#-0	379	$017B
G-0	401	$0191
G#-0	425	$01A9
A-0	451	$01C3
A#-0	477	$01DD
H-0	506	$01FA
C-1	536	$0218
C#-1	568	$0238
D-1	602	$025A
D#-1	637	$027D
E-1	675	$02A3
F-1	716	$02CC
F#-1	758	$02F6
G-1	803	$0323
G#-1	851	$0353
A-1	902	$0386
A#-1	955	$03BB
H-1	1012	$03F4
C-2	1072	$0430
C#-2	1136	$0470
D-2	1204	$04B4
D#-2	1275	$04FB
E-2	1351	$0545
F-2	1432	$0598

F#-2	1517	$05ED
G-2	1607	$0647
G#-2	1703	$06A7
A-2	1804	$070C
A#-2	1911	$0777
H-2	2025	$07E9
C-3	2145	$0861
C#-3	2273	$08E1
D-3	2408	$0968
D#-3	2551	$09F7
E-3	2703	$0A8F
F-3	2864	$0B30
F#-3	3034	$0BDA
G-3	3215	$0C8F
G#-3	3406	$0D4E
A-3	3608	$0E18
A#-3	3823	$0EEF
H-3	4050	$0FD2
C-4	4291	$10C3
C#-4	4547	$11C3
D-4	4817	$12D1
D#-4	5103	$13EF
E-4	5407	$151F
F-4	5728	$1660
F#-4	6069	$17B5
G-4	6430	$191E
G#-4	6812	$1A9C
A-4	7217	$1C31
A#-4	7647	$1DDF
H-4	8101	$1FA5
C-5	8583	$2187
C#-5	9094	$2386
D-5	9634	$25A2

D#-5	10207	$27DF
E-5	10814	$2A3E
F-5	11457	$2CC1
F#-5	12139	$2F6B
G-5	12860	$323C
G#-5	13625	$3539
A-5	14435	$3863
A#-5	15294	$3BBE
H-5	16203	$3F4B
C-6	17167	$430F
C#-6	18188	$470C
D-6	19269	$4B45
D#-6	20415	$4FBF
E-6	21629	$547D
F-6	22915	$5983
F#-6	24278	$5ED6
G-6	25721	$6479
G#-6	27251	$6A73
A-6	28871	$70C7
A#-6	30588	$777C
H-6	32407	$7E97
C-7	34334	$861E
C#-7	36376	$8E18
D-7	38539	$968B
D#-7	40830	$9F7E
E-7	43258	$A87A
F-7	45830	$B306
F#-7	48556	$BDAC
G-7	51443	$C8F3
G#-7	54502	$D4E6
A-7	57743	$E18F
A#-7	61176	$EEF8
H-7	64814	$FD2E

Peliohjaimen ja näppäimistön lukeminen

Pelaaja vaikuttaa kuvaruudun tapahtumiin yleensä joystickin tai näppäimistön välityksellä. Muita mahdollisia ohjaimia ovat paddle, valopistooli ja hiiri, mutta ne ovat kuusnelosessa harvinaisia. Silloin tällöin valmistajat ovat innostuneet tekemään monipuolisempia versioita perinteisestä joystickista, mutta ne ovat jääneet lyhytikäisiksi kokeiluiksi.

Joystick

Kuusneloseen voi liittää kaksi peliohjainta. Niiden lähettämät signaalit luetaan muistipaikoista $DC00 ja $DC01:

portti 2 -> $DC00
portti 1 -> $DC01

Porttien järjestys ei ole painovirhe. Jostain syystä peliportit on liitetty rekistereihin oudon takaperoisesti.

Muistipaikkojen viisi alinta bittiä vastaavat peliohjaimen eri asentoja:

Bitti 0 – ylös
Bitti 1 – alas
Bitti 2 – vasemmalle
Bitti 3 – oikealle
Bitti 4 – tulitusnäppäin

Biteissä on käänteinen logiikka, toisin sanoen lepotilassa kaikki bitit ovat ykkösiä. Jos painetaan tulitusnäppäintä tai väännetään kahvaa, vastaava bitti muuttuu nollaksi.

Peliohjainta on mukavampi lukea, kun käännetään kaikki bitit. Se käy helposti käskyllä EOR #$FF. Esimerkiksi seuraava rutiininpätkä tutkii kumpaan tahansa porttiin liitetyn peliohjaimen tilan:

```
JOYS    LDA $DC00    ;portti 2
        AND $DC01    ;lisätään portin 1 nollabitit
        EOR #$FF     ;käännetään bitit
        TAY
        AND #$10     ;bitti 4 X-rekisteriin
        TAX
        TYA
        AND #$0F     ;bitit 0-3 akkuun
        RTS
```

X-rekisterissä on tulitusnäppäimen tila. Jos se on erisuuri kuin nolla, näppäintä on painettu. Akussa palautettava luku tulkitaan seuraavan taulukon mukaan:

Peliohjaimen tila	Arvo	Bitit 0-3
lepotilassa	0	0000
ylös	1	0001
alas	2	0010
vasemmalle	4	0100
oikealle	8	1000
ylös & vasemmalle	5	0101
alas & vasemmalle	6	0110
ylös & oikealle	9	1001
alas & oikealle	10	1010

Erillisen lukurutiinin käyttäminen ei aina ole paras mahdollinen ratkaisu. Yhtä hyvin bitit voi tutkia suoraan silloin kun niitä tarvitaan, sillä tulos pitää joka tapauksessa muokata tarkoitukseen sopiviksi arvoiksi. Esimerkiksi kun ohjaillaan pelihahmoa vaakaan vierivällä ruudulla, vieritysnopeuden muuttamiseen sopivia arvoja ovat oikea = –1 ja vasen = 1. Toisin sanoen jos peliohjainta väännetään oikealle, ruutu vierii kiihtyvää vauhtia vasemmalle.

Näppäimistö

Ohjelmoijan kannalta olisi käytännöllisintä, jos näppäimistöä voisi lukea suoraan yhdestä muistipaikasta. Joissain mikroissa tämä onkin mahdollista, mutta ei kuusnelosessa. Näppäimistön lukemiseen tarvitaan kahta CIA-piirin rekisteriä:

portti A -> $DC00
portti B -> $DC01

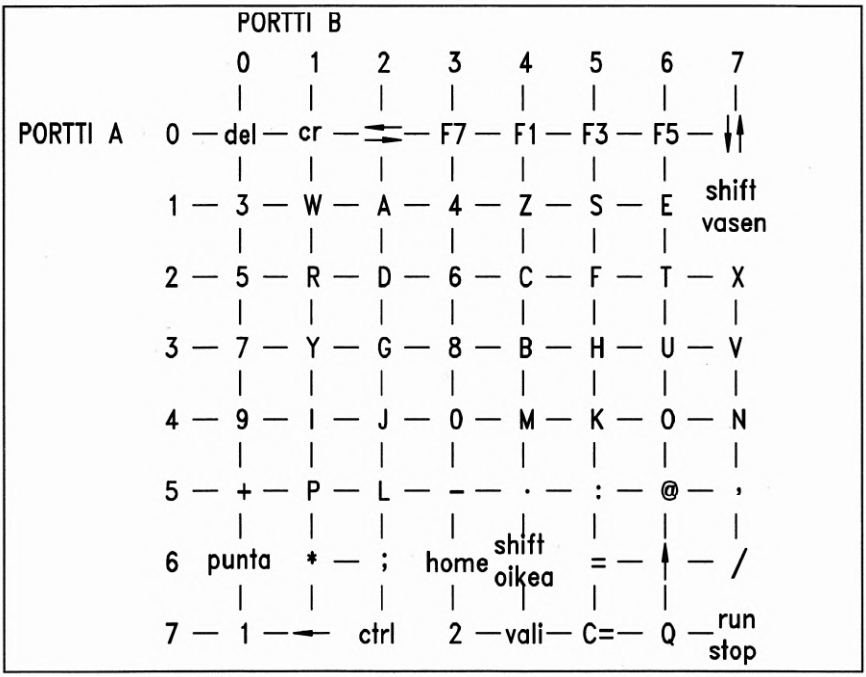
Muistipaikat muodostavat näppäimistömatriisin, jossa on kahdeksan riviä ja kahdeksan saraketta (kaavio 10.1). Käytössä on siis kaikkiaan kuusikymmentäneljä näppäintä. RESTORE-näppäintä ei voi lukea matriisista, sillä se on kytketty suoraan NMI-keskeytyslinjaan.

Portti A vastaa matriisin vaakariviä ja portti B saraketta. Näppäimistö tutkitaan matriisirivi kerrallaan kahdeksan näppäimen ryhmissä. Ensin on kirjoitettava vaakarivi muistipaikkaan $DC00. Tämän jälkeen muistipaikasta $DC01 voi tutkia, onko jotain kyseisen rivin näppäimistä painettu. Kummankin muistipaikan biteissä on käänteinen logiikka aivan samaan tapaan kuin peliohjaimia luettaessa.

Tutkitaan esimerkiksi näppäintä F1, joka sijaitsee matriisissa rivillä nolla ja sarakkeessa neljä. Aluksi muistipaikkaan $DC00 kirjoitetaan $FE (%11111110), siis bittiä nolla lukuunottamatta bitit ovat ykkösiä. Nyt muistipaikasta $DC01 tutkitaan bitti neljä. Mikäli se on nolla, näppäintä F1 on painettu. Rekisterin arvona ei välttämättä ole $EF (%11101111), sillä samaan aikaan on voitu painaa alas muitakin näppäimiä. Esimerkiksi arvo $E7 (%11100110) tarkoittaa, että on painettu näppäimiä F1, F7 ja DEL. Matriisista on siis mahdollista tutkia myös useiden näppäimien yhdistelmiä.

Luvun lopussa on esimerkkirutiini, jossa tutkitaan kaikki näppäimet. Tulos tallennetaan muistipaikoihin KEYVAL ja LASTK. Niiden arvo ilmoittaa näppäimen paikan matriisissa luettuna vasemmalta oikealle ja ylhäältä alas. Näin ollen DEL antaa tulokseksi nolla ja RUN/STOP kuusikymmentäkolme. Tulos $FF tarkoittaa, että näppäintä ei ole painettu.

Tuloksen antaminen kahdessa muistipaikassa johtuu siitä, että näppäimistöä voi lukea kahdella eri tavalla. Muistipaikkaan KEYVAL näppäinkoodi tallennetaan vain kerran yhtä painallusta kohden. Seuraavilla lukukerroilla arvoksi saadaan taas $FF, kunnes näppäin päästetaan välillä ylös tai painetaan jotain muuta näppäintä. Muistipaikasta LASTK voi puolestaan aina lukea, mitä näp-



Kaavio 10.1. Kuusnelosen näppäimistö on yhdistetty muistipaikkoihin $DC00 ja $DC01 8 X 8 linjan matriisina.

päintä pidetään alhaalla. Kirjoitettaessa tekstiä ruudulle on käytettävä muistipaikkaa KEYVAL, sillä muutoinhan ruudulle ilmestyisi pitkä rivi samaa kirjainta aina kun näppäintä painetaan. Jos taas näppäimistöä käytetään pelihahmon ohjailuun, näppäinkoodi kannattaa lukea muistipaikasta LASTK.

Käytännön ongelmat

Sekä näppäimistön että peliohjainporttien liittäminen samoihin muistipaikkoihin hankaloittaa näppäimistön tutkimista, varsinkin jos ohjelma tarvitsee sekä ohjaimia että näppäimistöä samaan aikaan. Porttiin 2 liitetty peliohjain häiritsee matriisin sarakkeen lukemista nollaamalla bittejä rekisterissä $DC00. Esimerkiksi tulitusnäppäin nollaa bitin neljä. Mikäli mitään näppäintä matriisin rivillä neljä ei ole painettu, ohjain ei aiheuta häiriöitä. Mutta jos painetaan samaan aikaan esimerkiksi näppäintä 9, lukurutiini antaa väärän tuloksen. Matriisin kolmea viimeistä riviä ohjain ei pysty sekoittamaan.

Porttiin 1 liitetty ohjain on vielä hankalampi tapaus, sillä se sotkee näppäimistön luvun lähes aina. Esimerkkirutiinissa ainoastaan matriisin alimman rivin kolme viimeistä näppäintä C=, Q ja RUN\STOP ovat luotettavia vaihtoehtoja. Jos kaikki tarvittavat näppäimet tutkitaan erikseen, on mahdollista käyttää matriisin kolmen viimeisen sarakkeen näppäimiä.

Näppäimistön ja molemmat peliohjaimet voi yrittää saada toimimaan rinnakkain lisäämällä lukurutiiniin moninkertaisia tarkistuksia. Kirjoittamalla aluksi rekisteriin $DC00 luku $FF on mahdollista lukea pelkästään peliohjainten nollaamat bitit. Tätä tietoa voi sitten käyttää hyväksi näppäimistön luvussa. Esimerkki: Portin 2 peliohjainta on väännetty ylös ja näppäimistömatriisissa sarakkeen kolme bitti oli rivistä riippumatta nolla. Peliohjaimen bitti vastaa matriisin riviä nolla. Näppäimen täytyi siis olla F7.

Täysin vedenpitäväksi näppäimistönlukua ei pysty tekemään, sillä aina on mahdollista, että peliohjainta väännetään huonoimmalla mahdollisella hetkellä lukurutiinin aikana. Esimerkkirutiinissa on eliminoitu ainoastaan porttiin 2 liitetyn ohjaimen aiheuttamat häiriöt. Näppäimistöä ei lueta lainkaan, mikäli rekisteristä $DC00 löytyy ylimääräisiä nollabittejä.

```
****************************************
;Näppäimistön luku

;KEYVAL = näppäinkoodi

;Näppäintä ei painettu:
;KEYVAL = $FF
;LASTK  = $FF

;Näppäin on sama kuin viimeksi:
;KEYVAL = $FF
;LASTK  = näppäinkoodi
****************************************
KEYSCAN LDA #$FF      ;oletus: ei näppäintä
        STA KEYVAL
        LDY #7        ;tutkitaan 8 matriisin riviä
L001    LDA ROWBIT,Y  ;kirjoitetaan rivin arvo
        STA $DC00
        CMP $DC00     ;pysyykö vakiona?
        BNE L002      ;ei pysy, kakkosportin
                      ;joystick häiritsemässä
        LDX $DC01     ;tarkistetaan sarakkeen bitit
        CPX #$FF
        BNE L003      ;kun arvo <> $FF, näppäintä
                      ;painettu
        DEY
        BPL L001      ;seuraava rivi
        LDA #$FF      ;ei näppäintä
        STA LASTK
L002    RTS
L003    STY ROW       ;rivi muistiin
        LDY #7        ;sarakkeen paikaksi tulee 0-7
L004    TXA           ;sarakkeen arvo X-rekisterissä
        AND COLBIT,Y  ;etsitään luvusta nollabitti
        BEQ L005      ;kun bitti = 0, Y-rekisterissä
                      ;sarake
        DEY
        BPL L004
```

```
L005    CLC
        LDX ROW          ;rivi
        TYA              ;sarake
        ADC ROWVAL,X     ;näppäinkoodi
                         ;= sarake + (rivi X 8)
        CMP LASTK
        BEQ L006
        STA KEYVAL       ;eri näppäinkoodi kuin
                         ;viimeksi
        STA LASTK        ;laitetaan muistiin
        RTS
L006    LDA #$FF         ;sama näppäinkoodi kuin
                         ;viimeksi
        STA KEYVAL       ;koodi vain muuttujassa LASTK
        RTS
COLBIT  BYTE $01,$02,$04,$08,$10,$20,$40,$80
ROWBIT  BYTE $FE,$FD,$FB,$F7,$EF,$DF,$BF,$7F
ROW     BYTE 0
ROWVAL  BYTE $00,$08,$10,$18,$20,$28,$30,$38
KEYVAL  BYTE $FF
LASTK   BYTE $FF
```

Hyödyllisiä Kernalin rutiineita

Basic-tulkista ja Kernalista löytyvät rutiinit ovat yleensä liian hitaita, jotta niistä olisi hyötyä pelien ohjelmoinnissa. On kuitenkin tiettyjä tehtäviä, joissa koneessa valmiina olevat rutiineita kannattaa käyttää.

Tiedoston lataus

Kernalin LOAD-rutiinin kutsuosoite on $FFD5. Ennen kuin sitä voi käyttää, on kutsuttava kahta valmistelevaa rutiinia, SETLFS ja SETNAM. Niiden avulla määritellään oheislaitteeseen liittyvät parametrit ja tiedoston nimi. Seuraava esimerkki havainnolistaa, miten ohjelmoidaan konekielinen vastine Basicin käskylle LOAD "nimi",8,1:

```
LATAUS LDA #$FF     ;tiedostonumero (ei merkitystä)
       LDX #8       ;laitenumero, 8 = levyasema
       LDY #1       ;osoitenumero, 1 = ei muuteta
       JSR $FFBA    ;SETLFS-rutiiniin
       LDA #4       ;nimen pituus
       LDX #<NIMI   ;nimen osoitteen alatavu
       LDY #>NIMI   ;nimen osoitteen ylätavu
       JSR $FFBD    ;SETNAM-rutiiniin
       LDA #0       ;0 = lataa, 1 = vertaa
       LDX #$FF     ;osoitteen alatavu
       LDY #$FF     ;osoitteen ylätavu
       JSR $FFD5    ;LOAD-rutiiniin
       RTS
NIMI   .BYTE "nimi"
```

Tiedostonumeroa ei latausrutiinissa tarvita. Sillä on merkitystä vain käytettäessä samaan aikaan useita levyaseman kanavia. Laitenumerolla on sama merkitys kuin LOAD-käskyssä, siis 1 tarkoittaa kasettiasemaa ja 8 levyasemaa. Osoitenumero on hyvin tärkeä. Jos se on erisuuri kuin nolla, tiedosto ladataan samaan osoitteeseen kuin mistä se on tallennettu. Jos se on nolla, alkusoite on

annettava erikseen. Basicin LOAD-käskyssä alkuosoitteeksi asetetaan aina $0801, mutta konekielellä osoite on annettava X- ja Y-rekisterissä ennen LOAD-rutiinin kutsumista. Esimerkissä osoitenumero on 1, joten alkuosoitetta ei tarvitse erikseen asettaa.

Tiedoston tallennus

Hyppyosoitteella $FFD8 kutsutaan Kernalin SAVE-rutiinia. Tallennus on valmisteltava SETLFS- ja SETNAM-rutiineilla aivan samaan tapaan kuin tiedoston latauksessa. Lisäksi tallennusrutiinille on kerrottava ohjelman alku- ja loppuosoite:

```
TALLET LDA #$FF     ;tiedostonumero (ei merkitystä)
       LDX #8       ;laitenumero, 8 = levyasema
       LDY #$FF     ;osoitenumero, (ei merkitystä)
       JSR $FFBA    ;SETLFS-rutiiniin
       LDA #4       ;nimen pituus
       LDX #<NIMI   ;nimen osoitteen alatavu
       LDY #>NIMI   ;nimen osoitteen ylätavu
       JSR $FFBD    ;SETNAM-rutiiniin
       LDA #<ALKU   ;alkuosoitteen alatavu
       STA $FB      ;osoittimen alatavu
       LDA #>ALKU   ;alkuosoitteen alatavu
       STA $FC      ;osoittimen ylätavu
       LDX #<LOPPU  ;loppuosoitteen alatavu
       LDY #>LOPPU  ;loppuosoitteen ylätavu
       LDA #$FB     ;nollasivun osoittimen osoite
       JSR $FFD8    ;SAVE-rutiiniin
       RTS
NIMI   .BYTE "nimi"
```

Ohjelman alkuosoite on tallennettava nollasivulle peräkkäisiin osoitteisiin. Esimerkissä tähän on käytetty muistipaikkoja $FB ja $FC.

Sekä latauksessa että tallennuksessa nollasivun muistipaikalla $9D on tärkeä merkitys. Jos sen sisältönä on #$80, ruudulle tulostetaan viesti "Searching for nimi" ja "Loading nimi" tai talletuksen yhteydessä "Saving nimi". Näiltä viesteiltä vältytään sijoittamalla muistipaikan arvoksi nolla.

Näppäimistön luku ja tekstin tulostus

Tekstiseikkailuissa ja muissa vastaavissa yhteyksissä näppäimistön lukeminen ja tekstin kirjoittaminen ruudulle eivät ole erityisen aikakriittisiä operaatioita. Tulostuksen voi aivan hyvin hoidella Kernalin rutiineilla ja samalla säästyy jonkin verran muistia.

Näppäimistön lukeminen ja tekstin kirjoittaminen ruudulle on hyvin yksinkertainen operaatio:

```
PRINT  JSR $FF9F    ;SCNKEY: tutki näppäimistö
       JSR $FFE4    ;GETIN: lue ASCII-koodi
       CMP #0       ;onko <> 0
       BEQ NOKEY
       JSR $FFD2    ;CHROUT: kirjoita ruudulle
NOKEY  RTS
```

Tekstin järjestelyä helpottaa Kernalin rutiini PLOT, jolla kursorin voi siirtää haluamaansa kohtaan:

```
SETCUR LDX #5       ;rivi 5
       LDY #0       ;sarake 0
       CLC
       JSR $FFF0    ;PLOT: aseta kursori
       RTS
```

Samaa rutiinia voi käyttää toisellakin tavoin. Jos ennen PLOT-rutiinia carry asetetaan päälle käskyllä SEC, tulokseksi saadaan kursorin paikka X- ja Y-rekistereissä.

Matematiikkaa

Peliohjelmoija tulee yleensä toimeen vaatimattomallakin matematiikan tuntemuksella. Poikkeuksena ovat lähinnä vektorigrafiikan tapaiset mutkikkaammat sovellutukset. On kuitenkin hyvä hallita tiettyjä perusasiota, kuten yhteen- ja vähennyslasku sekä kerto- ja jakolasku.

Negatiiviset luvut

Desimaalilukujen yhteydessä on totuttu siihen, että luvun edessä on erillinen miinusmerkki, jos luku on negatiivinen. Binaariluvuissa ongelma on ratkaistu siten, että luvun ylin bitti tulkitaan etumerkiksi. Jos bitti on päällä, luku on negatiivinen. Kahdeksanbittisistä luvuista 0–127 ($00–$7F) ovat positiivisia ja 128–255 ($80–$FF) negatiivisia. Vastaavasti kuusitoistabittisissä luvuissa $8000–$FFFF ovat negatiivisia.

Etumerkilliset binaariluvut kuitenkin poikkeavat desimaaliluista siinä, että pelkkä ylimmän bitin muuttaminen ei riitä. Esimerkiksi –1 ei suinkaan ole binaarina %10000001. Jos nollasta vähennetään yksi, tulokseksi saadaan %11111111 eli 255.

Binaarilukujen etumerkin kääntämisessä käytetään tekniikkaa, joka tunnetaan nimellä kahden komplementti. Se tarkoittaa sitä, että kaikki luvun bitit käännetään ja tulokseen lisätään yksi. Kokeillaanpa käytännössä, miten lasketaan kahden komplementti ykkösestä:

```
%00000001
%11111110    Käännetään bitit
%11111111    Lisätään yksi
```

Tulokseksi saatiin 255, kuten oli odotettavissa. Negatiiviset binaariluvut voivat aluksi vaikuttaa jotenkin nurinkurisilta, mutta systeemi on täysin looginen: –1 on suurin negatiivinen kokonaisluku ja toisaalta taas 255 on suurin kahdeksanbittinen luku.

Luku –128 eli %10000000 on siitä poikkeuksellinen, että sen etumerkkiä ei voi kääntää. Kahden komplementti luvusta on edelleen –128. Niinpä 127 (%01111111) on suurin mahdollinen positiivinen kahdeksanbittinen luku.

Etumerkillisiä binaarilukuja käytettäessä törmää usein tilanteeseen, jossa las-

kutoimituksen tuloksessa on väärä etumerkki. Esimerkiksi yhteenlaskun 127+1 tulokseksi saadaan –128, missä ei ole mitään järkeä. Prosessorin tilarekisterissä on ns. overflow- eli ylivuotobitti, joka varoittaa ohjelmoijaa siitä, että tulosta ei voi esittää seitsemällä bitillä. Ylivuodon voi tarkistaa ehdollisilla hyppykäskyillä BVC ja BVS.

Binaariluvut ovat siitä käteviä, että ne voi vapaasti tulkita joko etumerkillisiksi tai etumerkittömiksi. Kahdeksanbittisissä luvuissa on valittavana joko lukualue –128–127 tai 0–255.

Yhteen- ja vähennyslasku

Kaikki perus- tai kansakoulun käyneet varmasti muistavat, miten yhteenlaskuja tehtäessä luvut laskettiin yhteen numero kerrallaan ja yli menneet kymmenet laitettiin "muistiin". Kokeillaanpa tekniikan soveltamista binaarilukuihin:

```
    1    111      muistiin
    %10100111     (= 167)
+   %10000101     (= 133)
    %00101100     tulos = 44
```

Yhteenlaskun tulos on selvästi väärin. Tämä johtuu siitä, että viimeinen muistiin laitettu bitti ei enää mahdukaan yhteen tavuun. Oikea tulos olisi ollut 300, joten siitä puuttuu 256, mikä on juuri tuon yli jääneen bitin arvo.

Tätä tilannetta varten prosessorissa on ns. carry-bitti. Sillä on kaksi tehtävää. Toisaalta se ilmoittaa ohjelmoijalle, että tulos ei mahdu kahdeksaan bittiin ja samalla sitä voi käyttää muistibittinä laskettaessa yhteen kuusitoistabittisiä tai suurempi lukuja.

Edellä kuvattu laskutoimitus pitää suorittaa kahdessa osassa:

```
          0 <-- edellinen carry-bitti
    1    111      muistiin
    %10100111     (= 167)
+   %10000101     (= 133)
    %00101100     tuloksen alatavu = 44 ($2C)
```

```
          1 <-- edellinen carry-bitti
    %00000000
+   %00000000
    %00000001     tuloksen ylätavu = 1
```

Tuloksena on kuusitoistabittinen luku $012C, eli 300 desimaalina. Seuraavassa esimerkissä sama on toteutettu konekielellä:

```
ADD16  CLC
       LDA #167
       ADC #133
       STA LOBYTE
       LDA #0
       ADC #0
       STA HIBYTE
       RTS
```

Ensimmäinen yhteenlasku pistää carry-bitin päälle ja toinen yhteenlasku "lainaa" sitä lisäten tulosta yhdellä. Lainauksen jälkeen carry nollataan automaattisesti. Samalla tekniikalla voi laskea yhteen myös 24-bittisiä, 32-bittisiä tai vielä pidempiä lukuja.

Vähennyslaskussa on muuten sama periaate, mutta carry-bitillä on käänteinen merkitys. Tutkitaan esimerkiksi mitä tapahtuu, kun luvusta 256 ($0100) vähennetään yksi:

```
  1—11111112 lainataan carry (luvut desim.)
    %00000000
-   %00000001
    %11111111     tuloksen alatavu = 255 ($FF)

    %00000001
  0———>1    kun carry = 0, vähennetään lis. 1
-   %00000000
    %00000000     tuloksen ylätavu = 0
```

Ennen toista vähennyslaskua carry-bitti on nollattu, minkä seurauksena tulosta vähennetään yhdellä. Carryn on oltava päällä ennen laskujen aloittamista, sillä muutoin ensimmäisestä vähennyksestä saadaan väärä tulos. Lopussa carry-bitti

on aina automaattisesti ykkönen, mikäli lainausta ei tarvittu viimeisessä vähennyslaskussa. Alla on sama operaatio konekielisenä:

```
SUB16  SEC
       LDA #0
       SBC #1
       STA LOBYTE
       LDA #1
       SBC #0
       STA HIBYTE
       RTS
```

Yksi carry-bitin hienouksista on siinä, että sitä voi käyttää lukujen vertailussa. Nimittäin CMP, CPX ja CPY käskyt toimivat aivan kuin vähennyslasku, mutta ne eivät muuta vertailtavien lukujen arvoa ja carry-bittiä ei tarvitse laittaa päälle. Esimerkiksi jos CMP-käskyn jälkeen carry on pois päältä, akussa oleva luku on pienempi kuin luku johon sitä verrattiin.

Loogiset operaatiot: OR, AND ja EOR

Loogisilla operaatioilla tarkoitetaan kahden tai useamman luvun bittien yhdistämistä tiettyjen sääntöjen mukaisesti. Käytännön ohjelmoinnissa niille löytyy uskomattoman paljon käyttöä, joten niiden toimintaa kannattaa opiskella.

OR tarkoittaa suomennettuna "tai". Sen toimintaa voi kuvata seuraavasti: Jos bitti on ykkönen vähintään yhdessä luvussa, se on päällä myös lopputuloksessa. Kokeillaan käytännössä, miten kaksi lukua yhdistyvät OR-operaatiolla:

```
    %00001111     (= $0F)
OR  %00110011     (= $33)
    %00111111     (= $3F)
```

AND käännetään suomeksi "ja". Se kuvaakin osuvasti operaation logiikkaa: Jos bitti on päällä kaikissa luvuissa, se on päällä myös lopputuloksessa. AND-operaatio vaikuttaa lukujen bitteihin seuraavasti:

```
    %00001111     (= $0F)
AND %00110011     (= $33)
    %00000011     (= $03)
```

Logiikkaperheen mutkikkain operaatio on EOR, eli Exclusive Or. Sille on vaikea keksiä osuvaa suomennosta, ehkä lähimmäksi osuu "pois sulkeva tai". EOR on helpompi ymmärtää, kun sen määrittelee aluksi kahden luvun väliselle operaatiolle: Jos bitit ovat erisuuret, lopputuloksessa on ykkönen, muussa tapauksessa lopputuloksessa on nolla. Täydellisempi määritelmä kuuluisi: Jos bitti on ykkönen parittomassa määrässä lukuja, lopputuloksessa on ykkönen, muussa tapauksessa bitti on nolla. Toisin sanoen kaksi ykkösbittiä kumoavat toisensa:

```
    %00001111     (= $0F)
EOR %00110011     (= $33)
    %00111100     (= $3C)
```

EOR-operaatio on käytännöllinen silloin, kun on käännettävä tietyn bitin arvo. Tekemällä Exclusive Or luvun 255 kanssa, voidaan muuttaa kaikkien bittien arvot käänteisiksi:

```
    %00110011     (= $33)
EOR %11111111     (= $FF)
    %11001100     (= $CC)
```

Binaarikoodatut desimaaliluvut

Useimmissa prosessoreissa on erityinen desimaalimoodi. Kuusnelosessa se kytketään päälle käskyllä SED ja pois käskyllä CLD. Desimaali- eli BCD-moodissa lukuja käsitellään aivan kuin ne olisivat tavallisia desimaalilukuja. Esimerkiksi luku $19 tulkitaan normaalisti luvuksi 25, mutta desimaalimoodissa sen arvo on 19. Jos lukuun lisätään yksi, niin normaalisti tuloksena on 26 ($1A), mutta desimaalimoodissa 20:

```
      $19       BCD-arvo 19
+     $01       BCD-arvo 1
      $20       BCD-arvo 20
```

BCD-luvuissa lukualueena on periaatteessa 0–99, mutta joskus laskuissa voi käyttää myös desimaali- ja heksadesimaalien sekasotkua. Esimerkiksi luku $1F tulkitaan desimaalimoodissa luvuksi 25 (10+15). Laskutoimitusten yhdeydessä prosessori muuttaa tuloksen aidoksi desimaaliluvuksi, mikäli se on mahdollista:

```
      $1F       BCD-arvo 25
+     $01       BCD-arvo 01
      $26       BCD-arvo 26
```

Mikäli laskutoimituksessa syntyy kaksinkertainen ylivuoto, prosessori ei pysty käsittelemään sitä ja tuloksena on merkityksetön luku:

```
      $0F       BCD-arvo 15
+     $0B       BCD-arvo 11
      $10       BCD-arvo 10, oikea tulos olisi ollut 26
```

Kertolasku

Kertolaskun toteuttaminen konekielellä voi tuntua vaikealta ongelmalta, mutta ratkaisu on yllättävän yksinkertainen. Se on suoraan johdettavissa binaarilukujen logiikasta. Binaarilukujen bitit kuvaavat kahden potensseja, esimerkiksi 55 on binaarina %00110111, eli 32 + 16 + 4 + 2 + 1. Binaariluvuilla on sellainen ominaisuus, että jos sen bittejä siirretään kerran vasemmalle, tuloksena on luku kerrottuna kahdella. Kahden siirron jälkeen tuloksena on alkuperäinen luku kerrottuna neljällä, tai paremminkin kerrottuna kahdella kahteen kertaan. Jos taas bittejä siirretään nolla kertaa, tuloksena on tietysti luku itse.

Näitä tietoja soveltamalla kertolaskun voi pelkistää bittien siirroiksi ja yhteenlaskuiksi. Laskurutiinissa tutkitaan kertojan bitit yksi kerrallaan. Aina kun bitti on päällä, kerrottavaa siirretään bitin paikkaa vastaava määrä ja lisätään se tulokseen. Kokeillaan esimerkiksi kertoa kaksitoista viidellä (%00001100 X %00000101):

%00001100 kertojan bitti 0 päällä, nolla siirtoa (12 X 1 = 12)
+ %00110000 kertojan bitti 2 päällä, kaksi siirtoa (12 X 4 = 48)
%00111100 tulos: 12 + 48 = 60

Tehdään sama vielä toisin päin, siis kerrotaan viisi kahdellatoista:

%00010100 kertojan bitti 2 päällä, kaksi siirtoa (5 X 4 = 20)
+ %00101000 kertojan bitti 3 päällä, kolme siirtoa (5 X 8 = 40)
%00111100 tulos: 20 + 40 = 60

Esimerkkiin on tarkoituksellisesti valittu nelibittisiä lukuja, koska tällöin tulos mahtuu aina kahdeksaan bittiin. Yleensäkin tulosta varten on varattava kaksinkertainen määrä bittejä kuin mitä on kerrottavissa luvuissa.

Kahdeksanbittisiä lukuja kerrottaessa laskurutiinia voi nopeuttaa siten, että laitetaan sekä kertoja että tulos samaan kuusitoistabittiseen lukuun. Kertoja sijoitetaan alempaan tavuun ja sen bitit pyöritetään vasemmalta ulos. Samaan aikaan tulos pyöritetään sisään ylemmän tavun kautta. Oheinen esimerkki havainnollistaa ideaa. Kertojana on 55 ja kerrottavana 10:

%00001010 <——-kerrottava (8+2=10)
%00110111 <——-kertoja alussa (32+16+4+2+1=55)

%00001010	**%00011011**	**1. siirto, carry=1**
%00001111	**%00001101**	**2. siirto, carry=1**
%00010001	**%10000110**	**3. siirto, carry=1**
%00001000	**%11000011**	**4. siirto, carry=0**
%00001110	**%01100001**	**5. siirto, carry=1**
%00010001	**%00110000**	**6. siirto, carry=1**
%00001000	**%10011000**	**7. siirto, carry=0**
%00000100	**%01001100**	**8. siirto, carry=0**
%00000010	**%00100110**	**jaetaan tulos kahdella**

pyörityssuunta ————->

Tulokseksi saadaan 550 (512+32+4+2), aivan kuten pitikin. Aina kun alatavuun sijoitetun kertojan pyörittäminen pistää carry-bitin päälle, ylempään tavuun lisätään kerrottava. Lopussa tarvitaan yksi ylimääräinen pyöritys siksi, että kertojan bitit tutkitaan carry-bitin avulla. Niinpä kertoja on jatkuvasti

yhden pyörityksen tulosta edellä. Luvun lopussa on esimerkkirutiini, jossa on sovellettu tätä kertolaskusysteemiä.

Kuusitoistabittisten tai suurempien lukujen (**kuusitoistabittinen**)kertolaskua varten ei kannata kirjoittaa erillistä laskurutiinia, sillä toimituksen voi pilkkoa useaksi kahdeksanbittiseksi kertolaskuksi. Lukujen ala- ja ylätavut kertotaan keskenään, seuraavaksi kertotaan ala- ja ylätavut ristiin ja lopuksi tulokset lasketaan sopivasti yhteen:

```
     0  0 HI LO    luvun 1 alatavu X luvun 2 alatavu
    HI LO  0  0    luvun 1 ylätavu X luvun 2 ylätavu
     0 HI LO  0    luvun 1 alatavu X luvun 2 ylätavu
+    0 HI LO  0    luvun 1 ylätavu X luvun 2 alatavu
     32-bittinen tulos
```

Jakolasku

Jakolasku on huomattavasti vaikeampi hahmottaa kuin kertolasku. Yksinkertaisin tapa olisi kokeilla, kuinka monta kertaa jakajan voi vähentää jaettavasta, mutta tämä ei ole kovin käytännöllistä.

Jakolasku binaariluvuilla on tavallaan käänteinen operaatio kertolaskulle. Tutkitaan esimerkiksi jakolaskua 104/5:

%00000101 <——-jakaja (4+1=5)		
	%01101000 <——-jaettava alussa (64+32+8=104)	
%00000000	**%11010000**	**1. siirto, vertailuarvo 0**
%00000001	**%10100000**	**2. siirto, vertailuarvo 1**
%00000011	**%01000000**	**3. siirto, vertailuarvo 3**
%00000110	**%10000000**	**4. siirto, vertailuarvo 6 > 5**
%00000001		**vähennetään jakaja, carry = 1**
%00000011	**%00000001**	**5. siirto, vertailuarvo 3**
%00000110	**%00000010**	**6. siirto, vertailuarvo 6 > 5**
%00000001		**vähennetään jakaja, carry = 1**
%00000010	**%00000101**	**7. siirto, vertailuarvo 2**
%00000100	**%00001010**	**8. siirto, jakojäännös 4**
	%00010100	**9. siirto, tulos 20**
<———— pyörityssuunta		

Ideana on pyörittää jaettavan luvun bitit yksi kerrallaan vasemmanpuoleiseen tavuun. Kun siinä oleva luku on suurempi tai yhtäsuuri kuin jakaja, luvusta vähennetään jakaja ja carry-bitti muuttuu ykköseksi. Samalla kun jaettavaa pyöritetään vasemmalle, carry-bitti siirtyy tulokseen. Lopulta vasemmanpuoleisessa tavussa on jakojäännös ja oikeanpuoleisessa tulos. Luvun lopussa on tehokas konekielirutiini, joka toimii täsmälleen edellä kuvatulla periaatteella.

Jakolaskussa pahin ongelma on jakojäännös. Usein tarvitaan tarkkoja desimaalilukuja, varsinkin jos jakaja on suurempi kuin jaettava. Ongelman voi kiertää ainakin osittain soveltamalla eräänlaista "liukuvan pisteen" menetelmää. Jakojäännöksen tai jaettavan luvun bittejä siirretään vasemmalle, kunnes ylin bitti on päällä. Käytettyjen siirtokertojen määrä pistetään muistiin. Mikäli jakaja on vielä tämänkin jälkeen suurempi, sen bittejä siirretään puolestaan oikealle ja siirtokertojen määrää kasvatetaan. Tämän jälkeen suoritetaan jakolasku normaaliin tapaan. Lopuksi tulos on muokattava oikean suuruiseksi. Bitinsiirtoja vasemmalle on oltava yhteensä kahdeksan. Tehtyjen siirtojen määrä vähennetään kahdeksasta, jolloin tiedetään kuinka monta kertaa tuloksen bittejä on vielä siirrettävä.

Otetaan kokeeksi käsittelyyn edellä olleen esimerkin 104/5 jakojäännös:

%00000100	**jakojäännös = 4**
%10000000	**viisi siirtoa vasemmalle, tulos 128**
%00011001	**128/5 = 25, jakojäännös 3**
%11001000	**kolme siirtoa vasemmalle, tulos 200**

Jakolaskun 104/5 kokonaisosa on siis 20 ja desimaaliosa 200. Desimaaliosan arvo on hieman epätarkka, sillä lopussa tehtyjen bitinsiirtojen vuoksi kolmen alimman bitin sisältöä ei tiedetä. Oikea tulos olisi ollut 20+4/5 eli 20.8, kun taas konekielellä tulokseksi saatiin 20+200/256 eli noin 20.78. Käsittelemällä vielä jakolaskun 128/5 jakojäännös kolme ja liittämällä tulos desimaaliosan jatkeeksi, tarkaksi arvoksi saadaan 204. Näin päästiin on jo lähemmäs totuutta: 20+204/256 = 20.797. Täysin tarkkaa arvoa ei tässä tapauksessa edes voi laskea, sillä desimaaliosana on päättymätön jakso %1100110011001100 jne.

Kahdeksanbittistä suurempien lukujen jakolaskua ei voi pilkkoa osiin samaan tapaan kuin kertolaskussa. Ei ainakaan silloin, kun jakaja on suurempi kuin 255. Niinpä on joko tingittävä tarkkuudesta tai tehtävä erillinen jakolaskurutiini suuria lukuja varten.

Erikoisfunktiot

Peruslaskuja vaativampia erikoisfunktioita ovat esimerkiksi neliöjuuri ja trigonometriset funktiot. Ne on turvallisinta ohjelmoida käyttämällä valmiiksi laskettuja taulukoita. Jos ohjelma syystä tai toisesta vaatii tarkkojen arvojen laskemista, joudutaan soveltamaan polynomikaavoja, liukuvan pisteen eli floating point -lukuja ja noin yleensäkin ollaan eksytty varsin kauas käytännönläheisestä peliohjelmoinnista. Mutta näihin matemaattisiin kummajaisiin perehtyminen on silti erittäin suositeltaa ohjelmoinnin harjoittelua.

```
*************************************************
;Etumerkitön 8-bittinen kertolasku.
;Nopeudeltaan optimoitu versio.

;Rutiinia kutsuttaessa:
;akku = kertoja, X-rekisteri = kerrottava

;Rutiinin aikana:
;LO = kerrottava, HI = kertoja

;Tuloksena kuusitoistabittinen luku:
;LO = alatavu, HI = ylätavu
*************************************************
MULTIU STA LO
       TXA          ;Statusrekisterin reaktiota
                    ;kaivataan.
       BEQ ZERO     ;Nolla on käsiteltävä erikseen,
       DEX          ;sillä aluksi kerrottavasta
                    ;vähennetään yksi.
       STX HI
       LDA #0       ;Seuraa yhdeksän
                    ;bitinpyöritystä:
       LSR LO       ;Kertojan bitit pyöritetään
                    ;oikealta ulos.
       BCC SH1      ;Jos bitti oli päällä,
       ADC HI       ;lisätään kerrottava
                    ;(HI + 1, koska carry = 1).
SH1    ROR A        ;Tuloksen bitit pyöritetään
                    ;vasemmalta sisään,
```

```
        ROR LO      ;samalla kertoja oikealta ulos.
        BCC SH2     ;jne.
        ADC HI
SH2     ROR A
        ROR LO
        BCC SH3
        ADC HI
SH3     ROR A
        ROR LO
        BCC SH4
        ADC HI
SH4     ROR A
        ROR LO
        BCC SH5
        ADC HI
SH5     ROR A
        ROR LO
        BCC SH6
        ADC HI
SH6     ROR A
        ROR LO
        BCC SH7
        ADC HI
SH7     ROR A
        ROR LO
        BCC SH8
        ADC HI
SH8     ROR A
        STA HI
        ROR LO
        RTS
ZERO    LDA #0      ;Erikoistapaus: kerrottava = 0
        STA HI
        STA LO
        RTS
```

```
************************************************
;Etumerkitön 8-bittinen jakolasku.
;Nopeudeltaan optimoitu versio.

;Rutiinia kutsuttaessa:
;akku = jaettava, X = jakaja

;Tuloksena 8-bittinen luku + jakojäännös:
;JAETT = tulos, akku = jakojäännös
************************************************
DIVU    STA JAETT
        STX JAKAJA
        LDA #0
        ASL JAETT
        ROL A
        CMP JAKAJA
        BCC D1
        SBC JAKAJA
D1      ROL JAETT
        ROL A
        CMP JAKAJA
        BCC D2
        SBC JAKAJA
D2      ROL JAETT
        ROL A
        CMP JAKAJA
        BCC D3
        SBC JAKAJA
D3      ROL JAETT
        ROL A
        CMP JAKAJA
        BCC D4
        SBC JAKAJA
D4      ROL JAETT
        ROL A
        CMP JAKAJA
        BCC D5
```

```
        SBC JAKAJA
D5      ROL JAETT
        ROL A
        CMP JAKAJA
        BCC D6
        SBC JAKAJA
D6      ROL JAETT
        ROL A
        CMP JAKAJA
        BCC D7
        SBC JAKAJA
D7      ROL JAETT
        ROL A
        CMP JAKAJA
        BCC D8
        SBC JAKAJA
D8      ROL JAETT
        RTS
```

Erikoistehosteita videopiirillä

Kuusnelosen videopiirillä voi tehdä temppuja, joita ei mainita virallisissa käsikirjoissa. VIC-piiriin on jäänyt muutamia porsaanreikiä, joiden avulla ohjelmoija voi pakottaa sen tekemään mahdottoman tuntuisia asiota. Tehokeinoista näyttävimpiä ovat reunusten poistaminen ja kuvan alkukohdan muuttaminen.

Reunusten avaaminen

Kuvaruudun reunukset on mahdollista poistaa sekä sivuilta että ylä- ja alareunasta. Poistamisen jälkeen reunuksen alle sijoitetut spritet jäävät näkyviin. Tätä voi käyttää pelissä hyväkseen esimerkiksi sijoittamalla pistetaulut reunuksen kohdalle, jolloin kuvaruudulle jää enemmän tilaa.

Ylä- ja alareunuksen poistaminen perustuu sopivalla hetkellä tehtyyn kuvan kaventamiseen. Normaalisti alareunus alkaa rasteririviltä $FB, kun taas kavennetussa kuvassa se alkaa riviltä $F7. Jos kuva kavennetaan rivin $F7 jälkeen, mutta ennen riviä $FB, videopiiri ei enää ehdi piirtää reunuksen kavennettua osaa. Kun reunuksen palauttaa jälleen normaaliksi riveillä $FC–$FF, niin jostain syystä videopiiri ei ala piirtää reunusta lainkaan. Tuloksena on pystysuunnassa laajennettu kuva, jossa spritet voi sijoittaa myös reunusten kohdalle.

Avattujen reunusten kohdalla videopiirin on tehtävä jotain aikansa kuluksi. Niinpä se täyttää alueen videopankin viimeisellä tavulla, joka on normaalisti osoitteessa $3FFF (16383). Tämä muistipaikka on syytä nollata, jotta reunus ei näyttäisi raidalliselta.

Reunusten poistaminen kuvan sivuilta toimii samalla periaatteella: kavennetaan kuvaa vaakasuunnassa oikealla hetkellä ja palautetaan se miltei välittömästi normaaliksi. Ikävä kyllä operaatio on tehtävä jokaisella rasteririvillä ja lisäksi se on ajoitettava erittäin tarkasti. Muulle ohjelmalle jää niin vähän aikaa, että tekniikasta ei ole kovin paljoa käytännön hyötyä.

Kuvan siirto pystysuunnassa

Kuvaruudun joka kahdeksannella rasteririvillä videopiirin on päivitettävä eräitä sisäisiä osoittimia, jotta se osaisi etsiä muistista oikean grafiikan. Tätä ominaisuutta voi käyttää hauskaan kikkailuun. Pystyvierityksen avulla kuvan ensim-

mäistä riviä on mahdollista siirtää siten, että rasterin päivitys ei koskaan tavoita sitä. Videopiiri aloittaa kuvan päivityksen vasta kun hämäysoperaatio lopetetaan. Niinpä koko kuva siirtyy ruudulla alaspäin.

Kuvan yläpuolelle jäävälle tyhjälle alueelle videopiiri sijoittaa grafiikka-alueen viimeistä tavua, jota se näyttää käyttävän aina, kun se ei keksi muutakaan tekemistä. Luvun lopussa on esimerkki (rutiini 2), jossa tekniikkaa on käytetty kuvan vierittämiseen toistuvasti ylhäältä alas. Pienillä muutoksilla rutiinia voi käyttää myös kuvan katkaisuun, jolloin kuvan yläosa säilyy ennallaan. Kuvan siirto aloitetaan vasta kun rasterin päivitys on saapunut katkaisukohtaan ja muuttujan LINPOS alkuarvona on katkaisukohta plus yksi.

Kuvan siirto vaakasuunnassa

Kuvan ensimmäinen rivi on erittäin kriittinen kohta rasterin päivityksessä. Pystyvierityksen arvoa ei saisi missään tapauksessa muuttaa liian lähellä kuvan alkua. Mutta mehän emme varoituksista piittaa, vaan kokeilemme mitä tapahtuu.

Jos kuvaa vierittää ylös sopivasti ennen ensimmäistä riviä, videopiiri aloittaa välittömästi kuvan piirtämisen. Tämän seurauksena koko kuva vinksahtaa sijoiltaan vaakasuunnassa. Kuva siirtyy oikealle ja siirron pituus riippuu siitä, missä kohtaa rasteririviä kuvan päivitys oli menossa, kun kuvaa nykäistiin ylös Y-vierityksellä. Vasempaan reunaan ilmestyy puolestaan grafiikkaa, joka oli aiemmin kuvan oikessa reunassa, mutta se on siirtynyt alas yhden merkkirivin verran.

Ohjelmointikikan soveltaminen käytännössä on varsin vaikeaa, sillä ajoitus on tehtävä yhden kellojakson tarkkuudella, jotta kuvan alkukohdan voisi säätää haluamaansa merkkisarakkeeseen. Se on kuitenkin mainio keino ratkaista bittikartan vierityksen liittyvät ongelmat. Normaaleilla keinoilla kokonaista bittikarttaa on mahdoton vierittää nopeasti, sillä kuusnelosen teho ei mitenkään riitä kahdeksan kilon kokoisen alueen liikuttamiseen 50 kertaa sekunnissa.

Ohesta löytyvää esimerkkiä (rutiini 3) tutkimalla selviää, miten kuvan pyörittäminen toteutetaan käytännössä. Pelkästä pyörityksestä ei vielä ole paljoakaan iloa. Täyden kierroksen jälkeen kuva on palautettava alkukohtaan, minkä seurauksena se nytkähtää pystysuunnassa. Esimerkkirutiiniin on lisättävä uuden grafiikan siirto kuvan reunuksen alle ja kaksoispuskurointi. Tässä tapauksessa "reunus" tarkoittaa sitä kuvan merkkisaraketta, joka sattuu osumaan reunuksen alle. Uutta grafiikkaa on siirrettävä sekä näkyvään kuvaan että kaksoispuskuroinnissa käytettävään kakkoskuvaan, jonka voi kuvitella sijaitsevan yhden ruudun mitan päässä vierityksen suunnassa. Kaksoispuskurointi on välttämätöntä,

sillä kun kuva on kertaalleen pyöritetty ympäri, keinot loppuvat kesken. Niinpä otetaan näkyviin valmiiksi piirretty kuva vaihtamalla videopankkia. Sitten kierroksen voikin aloittaa alusta.

```
************************************************
;RUTIINI 1
;=========
;Kuvan laajentaminen pystysuunnassa.
;Laajentamisen jälkeen ylä- tai alareunuksen
;alle sijoitettu sprite on näkyvissä.
************************************************
ORG $C000
*  Alustetaan keskeytys.
START  SEI
       LDA #>IRQ    ;keskeytysvektori
       STA $FFFE
       LDA #<IRQ
       STA $FFFF
       LDA #$7F     ;IRQ-keskeytys pois
       STA $DC0D
       LDA #1       ;rasterikeskeytys päälle
       STA $D01A
       LDA #$F8     ;keskeytyskohdaksi $F8
       STA $D012
       LDA #$1B
       STA $D011
       LDA $DC0D
       CLI
L001   JMP L001
*  Pystyreunusten avaaminen.
*  Keskeytys rasteririvillä $F8.
IRQ    STA REMA+1   ;rekisterit talteen
       STX REMX+1
       STY REMY+1
       LDA $D019    ;kuitataan keskeytys
       STA $D019
       LDA #$13     ;kavennetaan kuva pystysuunnassa
```

```
        STA $D011       ;(24 riviä)
        LDA #$FC        ;odotetaan rasteririviä $FC
L002    CMP $D012
        BNE L002
        LDA #$1B        ;palautetaan kuvan korkeus
        STA $D011       ;normaaliksi (25 riviä)
REMA    LDA #0
REMX    LDX #0
REMY    LDY #0
        RTI
        END

*********************************
;RUTIINI 2
;========
;Kuvaruudun alkukohdan
;muuttaminen pystysuunnassa.
;(ns. "bad line" -vieritys)
*********************************
ORG $C000
LINPOS = $02            ;muuttuja rasteririviä varten
START  SEI              ;keskeytykset pois
L001   LDA #$2D         ;siirretyn kuvan 1. rivi
       STA LINPOS
L002   LDA $D011        ;ohitetaan rateririvit 255-312
       BMI L002
L003   LDA $D012        ;otetaan kuvaan 8 riviä
                        ;ennakkoa
       CMP #$2C
       BCC L003
       JSR TESTLINE     ;siirretään kuva
L004   LDA $D011        ;odotetaan kuvan loppua
       BPL L004
       LDA #$17         ;vakioidaan Y-vieritys
       STA $D011
       INC LINPOS       ;kuva yhtä riviä alemmas
       LDA LINPOS
```

```
        CMP #$FB        ;siirretään riville $FB asti
        BNE L002
        JMP L001        ;..ja aletaan alusta
*  Alirutiini, jossa kuvan alku siirretään
*  muuttujan LINPOS määräämälle rasteririville.
RLINE   LDA $D012
        TAX
        AND #$07        ;merkkirivin kohta
        TAY
        LDA YLINEVAL,Y ;luetaan taulukosta Y-vieritys
        STA $D011
L005    CPX $D012       ;odotetaan seuraavaa riviä
        BEQ L005
        LDA $D012
        CMP LINPOS      ;jatketaan riville LINPOS asti
        BCC RLINE
        RTS
YSCROL .BYTE $17,$10,$11,$12,$13,$14,$15,$16
        END

************************************************
;RUTIINI 3
;=========
;Bitmap-vieritys:
;alirutiinit kuvaruudun pyöritykseen.

;Grafiikka:  $6000-
;Värimuisti: $4000-
;Ohjelman kutsuosoite on kohdassa START.
;HUOM: ohjelman muuttaminen ennen JOY-rutiinia
;voi pilata ajoituksen.
************************************************
        ORG $C000
        IRQCNT = $02
* 40 hyppyvektoria viiverutiineihin
DELAYS .WORD DEL1+19,DEL2+19,DEL1+18,DEL2+18
        .WORD DEL1+17,DEL2+17,DEL1+16,DEL2+16
```

```
        .WORD DEL1+15,DEL2+15,DEL1+14,DEL2+14
        .WORD DEL1+13,DEL2+13,DEL1+12,DEL2+12
        .WORD DEL1+11,DEL2+11,DEL1+10,DEL2+10
        .WORD DEL1+09,DEL2+09,DEL1+08,DEL2+08
        .WORD DEL1+07,DEL2+07,DEL1+06,DEL2+06
        .WORD DEL1+05,DEL2+05,DEL1+04,DEL2+04
        .WORD DEL1+03,DEL2+03,DEL1+02,DEL2+02
        .WORD DEL1+01,DEL2+01,DEL1+00,DEL2+00
* Viiverutiinit:
* Rutiineita tarvitaan kaksi, koska konekielikäsky
* NOP  kestää kaksi kellojaksoa, mutta ohjelmassa
* tarvitaan  yhden kellojakson tarkkuus. Rutiinissa
* DEL1 on yksi kellojakso vähemmän kuin rutiinissa
* DEL2.
* $EA = NOP-käsky
DEL1    .BYTE $EA,$EA,$EA,$EA,$EA
        .BYTE $EA,$EA,$EA,$EA,$EA
        .BYTE $EA,$EA,$EA,$EA,$EA
        .BYTE $EA,$EA,$EA,$EA,$EA
        STX $D011
        JMP LOOPEX
DEL2    .BYTE $EA,$EA,$EA,$EA,$EA
        .BYTE $EA,$EA,$EA,$EA,$EA
        .BYTE $EA,$EA,$EA,$EA,$EA
        .BYTE $EA,$EA,$EA,$EA
        BIT $EA
        STX $D011
        JMP LOOPEX
*  Rasterikeskeytys kohdassa $28.
*  Ajoituksessa käytetyt käskyt:
*  NOP       = 2 kellojaksoa
*  BIT 1     = 3 ——""——-
*  LDA (0,X) = 6 ——""——-
IRQ     STA REMA+1      ;rekisterit talteen
        STY REMY+1
        STX REMX+1
        LDA $D019       ;kuitataan keskeytys
```

```
        STA $D019
        LDA (0,X)       ;8 kellojakson viive
        NOP             ;
        LDA #$2A        ;ajoitukseen käytettävät rivit
        LDX #$2B
L001    CMP $D012       ;odotetaan riviä $2A
        BNE L001
        LDY #10         ;54 kellojakson viive
L002    DEY             ;
        BNE L002        ;
        BIT 1           ;
        CPX $D012       ;odotetaan riviä $2B ja
                        ;korjataan ajoitus
        BEQ L003        ;jos rivi = $2B,
                        ;viive 13 jaksoa
        BIT $EA         ;jos rivi = $2C,
                        ;viive 12 jaksoa
        JMP L004        ;
L003    NOP             ;
        NOP             ;
L004    LDA (0,X)       ;
        JSR SETDEL      ;viiveen asetukseen
        LDA #%00110111 ;Y-vierityksekseksi 7
        STA $D011
        LDA SCROLL      ;X-vieritys
        STA $D016
        JSR JOY         ;peliohjain
REMA    LDA #0
REMX    LDX #0
REMY    LDY #0
        RTI
* Valitaan ajoitusviive ja siirretään kuvaruutua.
SETDEL LDA #%00110001 ;Y-vieritys = 1
        STA $D011
        LDA (0,X)       ;17 kellojakson viive
        LDA (0,X)       ;
        BIT 1           ;
        NOP             ;
```

```
        LDA XCNT        ;XCNT määrää kuvan alkukohdan
                        ;(0-39)
        ASL A
        TAX
        LDA DELAYS,X    ;luetaan hyppyvektori
                        taulukosta DELAYS
        STA DELPNT
        LDA DELAYS+1,X
        STA DELPNT+1
        LDX #%00110000 ;Y-vieritys = 0
        LDA #$2F
S001    CMP $D012       ;odotetaan riviä $2F
        BNE S001
        LDY #11         ;59 kellojakson viive
S002    DEY             ;
        BNE S002        ;
        BIT $EA         ;
        JMP (DELPNT)    ;hypätään viiveeseen
LOOPEX RTS
DELPNT .WORD DEL1       ;viiveen hyppyvektori
XCNT    .BYTE 0         ;kuvanruudun alkukohta (0-39)
* Luetaan ilotikku portista 1 ($DC01)
JOY     INC IRQCNT      ;reagoidaan tikkuun joka
                        ;4. keskeytyksessä
        LDA IRQCNT
        AND #3
        BNE J003
        LDX XSPEED      ;uusi vieritysnopeus
        LDA $DC00
        AND #12
        CMP #12
        BEQ J002
        AND #4
        BNE J001
        INX
        .BYTE $24       ;$24 + DEX = BIT $CA -käsky
J001    DEX
J002    STX XSPEED
```

```
J003    CLC             ;uusi vierityksen arvo:
        LDA SCROLL      ;(SCROLL + XSPEED) AND 7
        AND #7
        ADC XSPEED
        TAX
        AND #7
        ORA #$10
        STA SCROLL
        TXA             ;tutkitaan ylivuoto:
        AND #$F8        ;kuvaa siirretään,
        BMI J006        ;jos joku biteistä 4-7 on päällä
        BNE J004
        RTS
J004    LSR A           ;siirretään kuvaa oikealle
        LSR A
        LSR A
        CLC
        ADC XCNT
        CMP #40
        BCC J005
        SBC #40
J005    STA XCNT
        RTS
J006    LSR A           ;siirretään kuvaa vasemmalle
        LSR A
        LSR A
        ORA #$E0
        CLC
        ADC XCNT
        BPL J007
        ADC #40
J007    STA XCNT
        RTS
SCROLL .BYTE $10    ;X-vierityksen arvo + bitti 4
XSPEED .BYTE $FC    ;vieritysnopeus (alkuarvo -4)
* Ohjelman alku:
* Videopiirin ja keskeytyksen alustus.
```

```
START  SEI
       LDA #$35     ;kernal pois
       STA 1
       LDA #2       ;videopankki
       STA $DD00
       LDA #$08     ;väri- ja grafiikkamuisti
       STA $D018
       LDA #<IRQ    ;keskeytysvektori
       STA $FFFE
       LDA #>IRQ
       STA $FFFF
       LDA #$7F     ;CIA-keskeytys pois
       STA $DC0D
       LDA #1       ;videopiirin keskeytys päälle
       STA $D01A
       LDA #$28     ;keskeytyskohta $28
       STA $D012
       LDA #$1B
       STA $D011
       LDA $DC0D
       CLI
*  Minimoi ajoitusvirhe odottamalla riviä $27 ja
*   suorittamalla
*  32 NOP -käskyä
*  (virhettä korkeintaan 1 kellojakso).
MAIN   LDA $D011
       BMI MAIN
       LDA $D012
       CMP #$27
       BNE MAIN
       .BYTE $EA,$EA,$EA,$EA,$EA,$EA,$EA,$EA
       .BYTE $EA,$EA,$EA,$EA,$EA,$EA,$EA,$EA
       .BYTE $EA,$EA,$EA,$EA,$EA,$EA,$EA,$EA
       .BYTE $EA,$EA,$EA,$EA,$EA,$EA,$EA,$EA
       JMP MAIN
       END
```

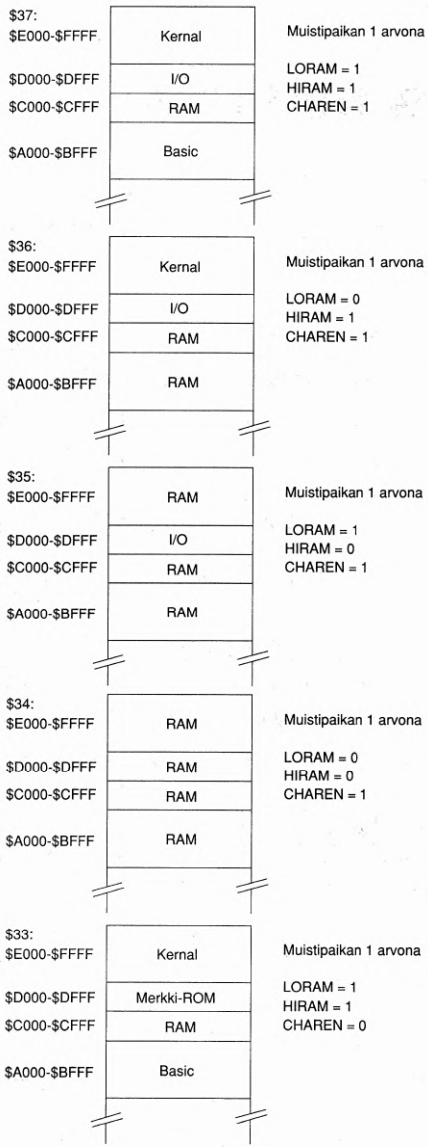
Liite 1
Muistin hallinta

Muistipaikan 1 avulla voi valita, mitkä osat muistilohkoista ovat RAM- ja mitkä ROM-muistia. Se myös kontrolloi I/O-piireille varatun muistialueen $D000-$DFFF toimintaa.

Bitti	Nimi	Merkitys
0	LORAM	Basic $A000-$BFFF: ROM tai RAM
1	HIRAM	Kernal $E000-$FFFF: ROM tai RAM
2	CHAREN	I/O lohko $D000-$DFFF: I/O, RAM tai merkkiROM
3		kasettiaseman kirjoituslinja
4		kasettiaseman nappulat, 0 = painettu
5		kasettiaseman moottori, 0 = päällä

Jos muistipaikan kolme alinta bittiä ovat päällä, Basic, Kernal ja I/O-piirit toimivat normaalisti. Muistipaikan arvona on yleensä $37, joskus käytetään arvoa $27, koska tällöin on mahdollista käyttää kasettiaseman kelaustoimintoa.

Muistilohkojen asettaminen RAMiksi tai ROMiksi EI tapahdu pelkästää nollaamalla tarvittavat bitit. Myös kaikkien kolmen bitin yhdistelmällä on merkitystä. Sen voi päätellä jo siitäkin, että I/O-lohkolla on kolme mahdollista tilaa, I/O, RAM tai merkkiROM. Seuraavalla sivulla on esimerkkejä yleisimmin käytetyistä muistiasetuksista.



Esimerkkejä yleisimmin käytetyistä muistiasetuksista.

Liite 2
Heksadesimaalit

$00	0	$A0	160	$1000	4096
$01	1	$B0	176	$2000	8192
$02	2	$C0	192	$3000	12288
$03	3	$D0	208	$4000	16384
$04	4	$E0	224	$5000	20480
$05	5	$F0	240	$6000	24576
$06	6			$7000	28672
$07	7	$0100	256	$8000	32768
$08	8	$0200	512	$9000	36864
$09	9	$0300	768	$A000	40960
$0A	10	$0400	1024	$B000	45056
$0B	11	$0500	1280	$C000	49152
$0C	12	$0600	1536	$D000	53248
$0D	13	$0700	1792	$E000	57344
$0E	14	$0800	2048	$F000	61440
$0F	15	$0900	2304		
		$0A00	2560	$10000	65536
$10	16	$0B00	2816		
$20	32	$0C00	3072		
$30	48	$0D00	3328		
$40	64	$0E00	3584		
$50	80	$0F00	3840		
$60	96				
$70	112				
$80	128				
$90	144				

Liite 3
Konekielikäskyt

* = Lisää yksi kellojakso, jos sivuraja ylittyy.

** = Lisää yksi kellojakso, jos haaraudutaan. Lisää kaksi kellojaksoa, jos haaraudutaan sivurajan yli.

Muistiosoitus	Käsky	Koodi	Kellojaksot
ADC	Lisää akkuun operandi + carry Jos tulos > 255, carry = 1, muutoin carry = 0		
suora	ADC #operandi	$69	2
nollasivu	ADC operandi	$65	3
nollasivu,X	ADC operandi,X	$75	4
absoluuttinen	ADC operandi	$6D	4
absoluuttinen,X	ADC operandi,X	$7D	4*
absoluuttinen,Y	ADC operandi,Y	$79	4*
(epäsuora,X)	ADC (operandi,X)	$61	6
(epäsuora),Y	ADC (operandi),Y	$71	5*
AND	Looginen AND akun ja operandin välillä		
suora	AND #operandi	$29	2
nollasivu	AND operandi	$25	3
nollasivu,X	AND operandi,X	$35	4
absoluuttinen	AND operandi	$2D	4
absoluuttinen,X	AND operandi,X	$3D	4*
absoluuttinen,Y	AND operandi,Y	$39	4*
(epäsuora,X)	AND (operandi,X)	$21	6
(epäsuora),Y	AND (operandi),Y	$31	5*

ASL Siirrä operandin bittejä vasemmalle
Operandin bitti 7 carryyn

akku	ASL A	$0A	2
nollasivu	ASL operandi	$06	5
nollasivu,X	ASL operandi,X	$16	6
absoluuttinen	ASL operandi	$0E	6
absoluuttinen,X	ASL operandi,X	$1E	7

BCC Haarautuminen, jos carry pois päältä

suhteellinen	BCC operandi	$90	2**

BCS Haarautuminen, jos carry päällä

suhteellinen	BCS operandi	$B0	2**

BEQ Haarautuminen, jos tulos nolla

suhteellinen	BEQ operandi	$F0	2**

BIT Bitit 6 ja 7 statusrekisteriin (V- ja N-liput)
Jos operandi AND akku on nolla, Z-lippu = 1

nollasivu	BIT operandi	$24	3
absoluuttinen	BIT operandi	$2C	4

BMI Haarautuminen, jos tulos negatiivinen

suhteellinen	BMI operandi	$30	2**

BEQ Haarautuminen, jos tulos erisuuri kuin nolla

suhteellinen	BNE operandi	$D0	2**

BEQ Haarautuminen, jos tulos positiivinen tai nolla

suhteellinen	BPL operandi	$10	2**

BRK Pakotettu keskeytys

sisäinen	BRK	$00	7

BVC Haarautuminen, jos tuloksessa ei ylivuotoa

suhteellinen	BVC operandi	$50	2**

BVS Haarautuminen, jos tuloksessa ylivuoto

suhteellinen	BVS operandi	$70	2**

CLC Nollaa carry-lippu

sisäinen	CLC	$18	2

CLD Nollaa BCD-tila (binaarikoodatut desimaalit)

sisäinen	CLD	$D8	2

CLI Mahdollista keskeytykset

sisäinen	CLI	$58	2

CLV Nollaa ylivuotolippu

sisäinen	CLV	$B8	2

CMP Vertaa operandia akkuun
Akku suurempi tai yhtäsuuri = C = 1
Akku pienempi = C = 0

suora	CMP #operandi	$C9	2
nollasivu	CMP operandi	$C5	3
nollasivu,X	CMP operandi,X	$D5	4
absoluuttinen	CMP operandi	$CD	4
absoluuttinen,X	CMP operandi,X	$DD	4*
absoluuttinen,Y	CMP operandi,Y	$D9	4*
(epäsuora,X)	CMP (operandi,X)	$C1	6
(epäsuora),Y	CMP (operandi),Y	$D1	5*

CPX Vertaa operandia X-rekisteriin
X-rekisteri suurempi tai yhtäsuuri = C = 1
X-rekisteri pienempi= C = 0

suora	CPX #operandi	$E0	2
nollasivu	CPX operandi	$E4	3
absoluuttinen	CPX operandi	$EC	4

CPY Vertaa operandia Y-rekisteriin
Y-rekisteri suurempi tai yhtäsuuri = C = 1
Y-rekisteri pienempi= C = 0

suora	CPY #operandi	$C0	2
nollasivu	CPY operandi	$C4	3
absoluuttinen	CPY operandi	$CC	4

DEC Vähennä operandia yhdellä

nollasivu	DEC operandi	$C6	5
nollasivu,X	DEC operandi,X	$D6	6
absoluuttinen	DEC operandi	$CE	6
absoluuttinen,X	DEC operandi,X	$DE	7

DEX Vähennä X-rekisteriä yhdellä

sisäinen	DEX	$CA	2

DEY Vähennä Y-rekisteriä yhdellä

sisäinen	DEY	$88	2

EOR Looginen EOR akun ja operandin välillä

suora	EOR #operandi	$49	2
nollasivu	EOR operandi	$45	3
nollasivu,X	EOR operandi,X	$55	4
absoluuttinen	EOR operandi	$4D	4

absoluuttinen,X	EOR operandi,X	$5D	4*
absoluuttinen,Y	EOR operandi,Y	$59	4*
(epäsuora,X)	EOR (operandi,X)	$41	6
(epäsuora),Y	EOR (operandi),Y	$51	5*

INC Lisää operandia yhdellä

nollasivu	INC operandi	$E6	5
nollasivu,X	INC operandi,X	$F6	6
absoluuttinen	INC operandi	$EE	6
absoluuttinen,X	INC operandi,X	$FE	7

INX Lisää X-rekisteriä yhdellä

sisäinen	INX	$E8	2

INY Lisää Y-rekisteriä yhdellä

sisäinen	INY	$C8	2

JMP Hyppää operandin määräämään osoitteeseen

absoluuttinen	JMP operandi	$4C	3
epäsuora	JMP (operandi)	$6C	5

JSR Hyppää alirutiiniin

absoluuttinen	JSR operandi	$20	6

LDA Siirrä operandi akkuun

suora	LDA #operandi	$A9	2
nollasivu	LDA operandi	$A5	3
nollasivu,X	LDA operandi,X	$B5	4
absoluuttinen	LDA operandi	$AD	4
absoluuttinen,X	LDA operandi,X	$BD	4*
absoluuttinen,Y	LDA operandi,Y	$B9	4*
(epäsuora,X)	LDA (operandi,X)	$A1	6
(epäsuora),Y	LDA (operandi),Y	$B1	5*

LDX Siirrä operandi X-rekisteriin

suora	LDX #operandi	$A2	2
nollasivu	LDX operandi	$A6	3
nollasivu,Y	LDX operandi,Y	$B6	4
absoluuttinen	LDX operandi	$AE	4
absoluuttinen,Y	LDX operandi,Y	$BE	4*

LDY Siirrä operandi Y-rekisteriin

suora	LDY #operandi	$A0	2
nollasivu	LDY operandi	$A4	3
nollasivu,X	LDY operandi,X	$B4	4
absoluuttinen	LDY operandi	$AC	4
absoluuttinen,X	LDY operandi,X	$BC	4*

LSR Siirrä operandin bittejä oikealle
Operandin bitti 0 carryyn

akku	LSR A	$4A	2
nollasivu	LSR operandi	$46	5
nollasivu,X	LSR operandi,X	$56	6
absoluuttinen	LSR operandi	$4E	6
absoluuttinen,X	LSR operandi,X	$5E	7

NOP Ei operaatiota (odota kaksi kellosykliä)

sisäinen	NOP	$EA	2

ORA Looginen OR akun ja operandin välillä

suora	ORA #operandi	$09	2
nollasivu	ORA operandi	$05	3
nollasivu,X	ORA operandi,X	$15	4
absoluuttinen	ORA operandi	$0D	4
absoluuttinen,X	ORA operandi,X	$1D	4*
absoluuttinen,Y	ORA operandi,Y	$19	4*

(epäsuora,X)	ORA (operandi,X)	$01	6
(epäsuora),Y	ORA (operandi),Y	$11	5*

PHA Kopioi akun sisältö pinomuistiin

sisäinen	PHA	$48	3

PHP Kopioi tilarekisteri pinomuistiin

sisäinen	PHP	$08	3

PLA Lue akun sisältö pinosta

sisäinen	PLA	$68	4

PLP Lue tilarekisterin sisältö pinosta

sisäinen	PLP	$28	4

ROL Pyöritä operandin bittejä vasemmalle
Carry siirretään operandin bittiin 0
Operandin bitti 7 siirretään carryyn

akku	ROL A	$2A	2
nollasivu	ROL operandi	$26	5
nollasivu,X	ROL operandi,X	$36	6
absoluuttinen	ROL operandi	$2E	6
absoluuttinen,X	ROL operandi,X	$3E	7

ROR Pyöritä operandin bittejä oikealle
Carry siirretään operandin bittiin 7
Operandin bitti 0 siirretään carryyn

akku	ROR A	$6A	2
nollasivu	ROR operandi	$66	5
nollasivu,X	ROR operandi,X	$76	6
absoluuttinen	ROR operandi	$6E	6
absoluuttinen,X	ROR operandi,X	$7E	7

RTI Paluu keskeytyksestä

sisäinen	RTI	$40	6

RTS Paluu alirutiinista

sisäinen	RTS	$60	6

SBC Vähennä akusta operandi
Jos carry = 0, tuloksesta vähennetään yksi
Jos akku operandi, carry = 0, muutoin carry = 1

suora	SBC #operandi	$E9	2
nollasivu	SBC operandi	$E5	3
nollasivu,X	SBC operandi,X	$F5	4
absoluuttinen	SBC operandi	$ED	4
absoluuttinen,X	SBC operandi,X	$FD	4*
absoluuttinen,Y	SBC operandi,Y	$F9	4*
(epäsuora,X)	SBC (operandi,X)	$E1	6
(epäsuora),Y	SBC (operandi),Y	$F1	5*

SEC Aseta carry-lippu

sisäinen	SEC	$38	2

SED Aseta BCD-tila (binaarikoodatut desimaalit)

sisäinen	SED	$F8	2

SEI Estä keskeytykset (ei päde NMI-keskeytyksiin)

sisäinen	SEI	$78	2

STA Kopioi akku operandin määräämään muistipaikkaan

nollasivu	STA operandi	$85	3
nollasivu,X	STA operandi,X	$95	4
absoluuttinen	STA operandi	$8D	4
absoluuttinen,X	STA operandi,X	$9D	4

absoluuttinen,Y	STA operandi,Y	$99	4
(epäsuora,X)	STA (operandi,X)	$81	6
(epäsuora),Y	STA (operandi),Y	$91	5

STX Kopioi X-rekisteri operandin määräämään muistipaikkaan

nollasivu	STX operandi	$86	3
nollasivu,Y	STX operandi,Y	$96	4
absoluuttinen	STX operandi	$8E	4

STY Kopioi Y-rekisteri operandin määräämään muistipaikkaan

nollasivu	STY operandi	$84	3
nollasivu,X	STY operandi,X	$94	4
absoluuttinen	STY operandi	$8C	4

TAX Kopioi akku X-rekisteriin

sisäinen	TAX	$AA	2

TAY Kopioi akku Y-rekisteriin

sisäinen	TAY	$A8	2

TSX Kopioi pino-osoitin X-rekisteriin

sisäinen	TSX	$BA	2

TXA Kopioi X-rekisteri akkuun

sisäinen	TXA	$8A	2

TXS Kopioi X-rekisteri pino-osoittimeen

sisäinen	TXS	$9A	2

TYA Kopioi Y-rekisteri akkuun

sisäinen	TYA	$9A	2

A

A-ajastin 23
Aaltomuoto 70
ADSR 72
Ajastimet 24
Alirutiini 18
Ammus 58
AND 98
Apuohjelma 15
Assemblerkääntäjä 14
Attack 72
Äänenvoimakkuus 70, 76
Äänikanava 70
Äänirekisterit 70

B

B-ajastin 23
Bad line 31
Band-pass-suodatin 74
Basic 117
Basic-tulkki 91
BCD-luvut 99
Binaarikoodatut desimaaliluvut 98
Bittigrafiikka 41
Bittikartta 41, 59
Bittikartan vieritys 108
Bittipari 40, 51
BREAK-käsky 20
BREAK-lippu 20

C

Carry-bitti 95
Chorus-efekti 74
CHROUT 93
CIA-piiri 22
Complex Interface Adapteri 22

D

Decay 72
Desimaalimoodi 98
Digitoitu musiikki 25
Digitoitu ääni 76

E

EOR 98
Etumerkki 94
Etumerkilliset binaariluvut 94
Exclusive Or 98

G

GETIN 93
Grafiikkamerkit 35
Grafiikkamuisti 42
Grafiikkapala 46
Grafiikkapuskuri 64

H

Haamurekisterit 53, 70
High-pass-suodatin 74
Hiiri 84
Hitausvoima 56
Hyppyvektori 20, 21, 26

I

I/O-piirit 36, 117
Inertia 56
Interrupt Request 20
IRQ-keskeytys 20

IRQ-rutiinit 21
IRQ-vektori 22
Isometrinen viisto-perspektiivi 45

J

Jaettava 102
Jakaja 102
Jakojäännös 102
Jakolasku 101
Joystick 84

K

Kahdeksansuuntainen vieritys 63
Kahden komplementti 94
Kaksoispuskurointi 64
Katkaisutaajuus 74
Kavennettu kuva 61
Kellojakso 24, 31
Kellotaajuus 24
Kernal 21, 117,91
Kerrottava 100
Kertoja 100
Kertolasku 99
Keskeytyksen alustus 21
Keskeytysten kontrolli-rekisteri 23
Keskeytyksen kuittaaminen 22
Keskeytykset 20
Keskeytyskohta 28, 65
Keskeytyslähde 20, 23
Keskeytyspyyntö 20, 22, 28
Keskeytysrutiini 20, 65
Keskeytystaajuus 24
Kohina 71
Kolmioaalto 71
Konekielimonitori 14
Kursori 93
Kuusitoistabittinen kertolasku 101
Kuvan jakaminen 30
Kuvan katkaisu 108
Kuvan pyörittäminen 108
Kuvan päivitystaajuus 24, 27
Kuvan siirto 107
Kuvaruudun reunukset 107
Kuvaruutukoodi 39
Kuvaruutumuisti 35, 42

L

Laajennettu kuva 107
Laajennettu taustaväritila 41
Laajennetut spritet 52
Laitenumero 91
Latausrutiini 91
Liukuva piste 102
LOAD 91
Loogiset operaatiot 97
Low-pass-suodatin 74

M

Maski 57
Merkkigrafiikka 35, 39, 56
Merkkimuisti 39
MerkkiROM 36, 48
Monivärigrafiikka 40, 42
Moniväriset merkit 40
Muistibitti 95
Muistilohkot 117
Multicolor mode 40

N

Negatiiviset luvut 94
Neliöaalto 71

NMI-keskeytykset 25, 77
NMI-vektori 25
Näppäimistö 86
Näppäimistömatriisi 86
Näppäimistön lukeminen 93
Näytteenottotaajuus 76

O

Oktaavi 74
Ongelmapelit 13
OR 97
Osoitenumero 92

P

Paddle 84
"Pahat linjat" 31
Parallaksiefekti 30
PD Systems 15
Pelikenttä 46
Peliohjain 84
Peliportit 84
Pikseli 34
Pino 20
PLOT 93
Polynomikaavat 103
Portamento 74
Portti 1 84
Portti 2 84
Portti A 86
Portti B 86
Pulssiaalto 71
Pulssin leveys 71
Pystyvieritys 60, 62

R

Rakenteinen ohjelmointi 18
RAM 117
Rasterikello 31
Rasterikeskeytys 24, 27, 28
Rasterirekisteri 27
Rasterirekisterin ylivuotobitti 27
Rasteririvi 27
Rasterivertailu 27
Read only -rekisteri 75
Release 72
Rengasmodulaatio 75
RESET-vektori 26
Resoluutio 34
Resonanssi 75
Restore-näppäin 25, 86
Reunus 61
Reunuksien päivitys 63
Reunusten avaaminen 107
Ristiinkääntäjä 15
ROM 117
Roolipelit 11
RTI-käsky 20

S

Saha-aalto 71
Samplaaminen 76
Sample 76
Satunnaisgeneraattori 71
Satunnaisluku 76
SAVE 92
SCNKEY 93
SEI-käsky 21
Seikkailupelit 10
Shoot'em up Construction Kit 15
SID 70
Simulaatiot 13
Siniaalto 71, 76
Sinikäyrä 70
Software-spritet 58

Sound Interface Device 70
Sprite 31, 47
Spriteanimaatio 49
Sprite-lohko 47
Sprite-osoittimet 48
Spritekeskeytykset 31
Spriten kytkeminen päälle 49
Spriten liike 55
Spriten liikeradat 56
Spriten prioriteetit 51
Spriten sijainti 49
Spriten törmäykset 30
Spriten värirekisteri 50
Spriten X-koordinaatti 49
Spriten Y-koordinaatti 49
Strategiapelit 12
Suodatin 74
Sustain 72
Sustain-taso 72
Synkronointi 75

T

Taajuusalue 74
Tallennusrutiini 92
Tarkkuusgrafiikka 40, 42
Tasavireinen asteikko 74
Taulukko 19
Taustagrafiikka 45
Taustaväri 34
Tekstiseikkailut 93
Tiedoston lataus 91
Tiedoston tallennus 92
Tiedostonumero 91
Tilarekisteri 20
Toimintapelit 9
Törmäysrekisteri 52

V

Vaakavieritys 60, 62, 65
Valokynä 31
Valopistooli 31, 84
Vapaasuuntainen vieritys 63
Vektorigrafiikka 45
Verhokäyrä 72
Vibrato 74
VIC 34, 107
Video Inferface Chip 34
Videopankki 35, 48
Videopiiri 35
Vieritys 60
Vieritysnopeus 61
Vieritysrekisteri 60
Vuokaavio 16
Vähennyslasku 96
Värikoodi 34, 40
Värimoodi 40
Värimuisti 35, 40
Värit 34
Väylä 31

W

Write only -rekisteri 70

Y

Yhteenlasku 95
Yksivärinen sprite 47
Ylivuotobitti 95
Yläsävelsarja 71

Commodore 64 on maailman levinneimpiä tietokoneita. Siinä yhdistyvät ihanteellisella tavalla monipuoliset grafiikka- ja ääniominaisuudet sekä ohjelmoinnin helppous.

Tämän oppaan avulla saat koneestasi entistä enemmän irti. Se selostaa perinpohjin kaikki peliohjelmoinnin kannalta olennaiset asiat. Miten tehdään kuvan sulava vieritys? Kuinka hallitaan keskeytyksiä? Miten grafiikka- ja äänipiiriä ohjelmoidaan? Lisäksi kirjassa on runsaasti assemblerkielisiä esimerkkirutiineita, joita tutkimalla opit nopeasti kuusnelosen salaisuudet.

Kirjoittajalla on useiden vuosien kokemus Commodore 64:n ohjelmoinnista ja kolme hänen tekemäänsä peliä on päässyt kansainväliseen levitykseen. Nyt sinullakin on mahdollisuus yltää samaan. Tässä oppaassa on yksissä kansissa kaikki tietous, mitä tarvitset kansainvälistä tasoa olevien pelien ohjelmointiin.